NAVAL POSTGRADUATE SCHOOL
Monterey, California

UNITED STATES NAVY · NAVAL POSTGRADUATE SCHOOL ·

THESIS

B45293

REAL TIME IMAGING AND INFRARED BACKGROUND SCENE ANALYSIS USING THE NAVAL POSTGRADUATE SCHOOL INFRARED SEARCH AND TARGET DESIGNATION (NPS-IRSTD) SYSTEM

by

Jean Daniel Bernier

September, 1991

Thesis Advisor:	Alfred W. Cooper
Co-Advisor	Ron Pieper

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION PMS-421 and Naval Postgraduate School Direct funded research; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
REAL TIME IMAGING AND INFRARED BACKGROUND SCENE ANALYSIS USING THE NAVAL POSTGRADUATE SCHOOL INFRARED SEARCH AND TARGET DESIGNATION (nps-irstd) SYSTEM

12. PERSONAL AUTHOR(S) Jean Daniel Bernier

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	1991, September	105

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	IRSTD, Framegrabber, Thermal Imaging, 386 Protected Mode

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The imaging in real time of infrared background scenes with the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System was achieved through extensive software developments in protected mode assembly language on an Intel 80386 oomputer.

The new software processes the 512 by 480 pixel images directly in the extended memory area of the computer where the DT-2861 frame grabber memory buffers are mapped. Direct interfacing, through a JDR-PR10 prototype card, between the frame grabber and the host computer AT bus enables each load of the frame grabber memory buffers to be effected under software control. The protected mode assembly language program can refresh the display of a six degree pseudo-oolor seotor in the scanner rotation within the two second period of the soanner.

A study of the imaging properties of the NPS-IRSTD is presented with preliminary work on image analysis and contrast enhancement of infrared background scenes.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL A.W. Cooper	22b. TELEPHONE (Include Area Code) 408-646-2452; 22c. OFFICE SYMBOL PH/Cr

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 Unclassified

Real Time Imaging and Infrared Background Scene Analysis using the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System

by

Jean Daniel Bernier
Major, Canadian Forces
BEng, Royal Military College of Canada, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

ABSTRACT

The imaging in real time of infrared background scenes with the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System was achieved through extensive software developments in protected mode assembly language on an Intel 80386 33 Mhz computer.

The new software processes the 512 by 480 pixel images directly in the extended memory area of the computer where the DT-2861 frame grabber memory buffers are mapped. Direct interfacing, through a JDR-PR10 prototype card, between the frame grabber and the host computer AT bus enables each load of the frame grabber memory buffers to be effected under software control. The protected mode assembly language program can refresh the display of a six degree pseudo-color sector in the scanner rotation within the two second period of the scanner.

A study of the imaging properties of the NPS-IRSTD is presented with preliminary work on image analysis and contrast enhancement of infrared background scenes.

345293
C.1

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I wish to thank my wife Margaret for the support she has given me and for her patience during the long months I have spent working on this project. She gave me the strength to carry on and friendly advise when I was frustrated. Without her help and care this project would not have been completed.

I also wish to express my gratitude to my advisor, Prof. Alfred W. Cooper for his guidance and for his trust in my abilities. He has taken the time to be my mentor and has taken interest not only in my work on this thesis but also in my progress and development as I strived to become a scientist. It is on his recommendation that I became a member of Sigma Xi and I thank him for the support and encouragement he has provided.

I wish to thank Mr. W.J. Lentz for all the time and effort he has invested in support of my thesis. Jerry has gone out of his way on numerous occasions in order to make my life easier so that I can bring this thesis to completion. This I will never forget and I hope we will keep in touch and work together again in the future.

Finally I wish to thank Ms. Ludye Oppel-Block for taking a keen interest in my work and for her support and assistance. I would have gone without eating on numerous occasions was it not for her taking such good care of me.

This project was funded jointly by the Navy Electro-Optic Projects Office (PMS-421), and the Naval Postgraduate School Direct Funded Research Program.

I. INTRODUCTION

A. RESEARCH OBJECTIVES

The NPS-IRSTD is a locally modified version of the Advanced Demonstration Model (ADM) of the Navy AN/SAR-8 infrared search and track system. It is a passive scanning surveillance system which provides a 360 degree coverage of the horizon. As a prototype for the Navy AN/SAR-8, the former ADM was designed to provide naval platforms with the capability of passively detecting, identifying and tracking multiple high-speed, low-flying targets in a maritime environment. Although the actual deployment of a surface IRST in one form or another is still only under assessment, the need for such a device to supplement other target detection and tracking systems on board naval sea and air platforms is widely recognized. [Ref. 8]

An objective actively pursued since 1989 at the Naval Academic Center for Infrared Technology (NACIT) is to display Infrared Search and Track (IRST) data in real time. This has been the object of two MS-theses [Refs. 2 and 3] prior to this one and preliminary results have already been presented in a number of papers [Refs. 5 and 6]. The ADM was designed for the detection and tracking of targets with low radar cross-sections and sea skimming flight profiles. For this reason, the ADM detector arrays, preamplifiers and optical assembly were designed to maximize the sensitivity of detection of small angular extent targets at the expense of imaging resolution [Ref. 6]. Nevertheless, the ability to display real time IRSTD output data in image form is motivated by the need to acquire and study the radiance characteristics of background and target scenes with a goal to evaluate target discrimination techniques, clutter suppression algorithms and range estimation methods, and to develop matched spatial frequency filters for target detection, identification and tracking [Ref 8]. The collection of both raw video and processed image files into a large IRSTD database is under way and will provide an invaluable source of experimental data for the statistical analysis of sea, air and land backgrounds in a variety of atmospheric conditions. In addition to background measurement and analysis, proposals for research in the near future include the study of two-color range estimation and target discrimination, and polarization effects in target discrimination as well as an extensive comparison between IRST imaging and data from a number of other electro-optic sensors [Ref. 8]. Specifically, comparative studies of scene statistics between the NPS-IRSTD and the AGA-780 Thermovision Camera are already in progress.

The NPS-IRSTD constitutes an ideal test bed for gathering raw IRST output data and supports the role of NACIT in the continuing development and assessment of IRST technology.

B. NPS-IRSTD SYSTEM OVERVIEW

1. System Description

The NPS-IRSTD is a passive infrared surveillance system which scans the horizon with two parallel vertical arrays of 90 Indium Antimonide (InSb) detectors operating in the 3 to 5 micrometer range. A full field of view of 360 degrees by 10.5 degrees (10 degrees above the horizon) is achieved through the rotation of the scanner assembly. The rotating Scanner Assembly consists in the following components: the optics, detector arrays, cryogenic cooler, Preamplifier Bandpass Assemblies (PBA), analog multiplexers, slip rings and position-in-rotation signal generating hardware.

At the heart of the optical assembly is a 14 inch Schmidt F/1 telescope with a 10 inch aperture aspherical corrector plate made of Germanium. The lead and lag arrays, displaced by one-half degree, are suspended from the top of the telescope housing and are positioned in the focal plane of the Schmidt telescope. The two arrays operate independently and are covered by filters which pass selected bands of IR radiation in the 3 to 5 micrometer range. Each detector is 0.02 inches high and 0.003 inches wide and has an instantaneous field of view of 2.0 by 0.3 milliradians [Ref 1].

The 90 detector channels are connected to six analog multiplexers each handling 15 detector channels. The outputs from the multiplexers come out of the scanner assembly through a set of slip rings at the base of the rotating head and are routed via six analog lines driven by the buffer power unit to the equipment room where they directly feed six analog to digital converters (ADCs).

The scanning assembly rotational position and the infrared scene data are correlated by means of two position-in-rotation signals. These signals are generated by a gear-driven optical disc position sensor mounted on the base of the scanner assembly giving 60,000 output pulses per revolution with an additional end-of-rotation (EOR) marker.

The six analog data channels coming out of the scanner are digitized by six 8-bit A/D converters and multiplexed by a digital multiplexer. Each detector is sampled

60,000 times per revolution. The rotation period of the scanner was increased by 8 percent to 2.16 seconds to produce an output data rate from the 180 detectors of 5.0 Megabytes per second (assuming that the lead array would eventually be operational).

The data is transferred via a coaxial cable to an HBR 3000i twelve channel tape recorder for storage. A unique 5.0 MHz clock crystal drives the multiplexers in the scanner assembly, the A/D converters and digital multiplexer as well as all the data latching circuitry for the tape recorder. A phase-locked loop is necessary for data recording because the clock input to the tape recorder must be exceptionally stable and symmetric. Another timing signal of importance is obtained by dividing the fundamental clock frequency by 180. The 1/180 pulse thus generated coincides with the sampling of the first detector in the lead array and is recorded along with the data on a separate channel. This pulse is used in the imaging process to indicate the top of the image. In short, four timing signals are recorded along with the data; these are the fundamental clock, the position-in-rotation pulses, the end-of-rotation pulses and the 1/180 clock [Ref 6].

2. Summary of past developments

The ADM received by NACIT in 1985 at the end of a period of sea trials was inoperable. The scanner assembly and buffer power unit were reassembled, refurbished and mounted on the roof of Spanagel Hall after the roof was cut off and replaced with a reinforced concrete structure. The original Data Conditioner Unit (DCU) and power control panels were installed in a room on the seventh floor of Spanagel. The rest of the signal processing equipment including an HBR 3000i twelve channel tape recorder, and a Masscomp 530 computer with an Optimem 1000 optical disc drive were positioned on the second floor.

The defective Stirling pump cryogenic system was replaced with a liquid nitrogen cooling reservoir by Crittenden and Cooper [Ref 4]. At the same time, the vacuum dewar insulating system was replaced with a foam insulation system. This modification necessitated changes to the optical system. The optical redesign for the removal of a dewar window was made by Parker [Ref 9] using the Super-Oslo code. The optical system was corrected and the individual detectors were calibrated by Ayers [Ref 1].

The original ADM was designed to provide a YUK computer with tracking information based on a reduced set of potential target data. A Background Normalizer Unit (BNU) in the scanner assembly accepted the signals coming from the detector pramplifier bandpass assemblies (PBAs) and provided suppression of all data patterns not meeting predefined criteria. However this circuitry limited the imaging capability and reduced the universality of the data for general use. In order to facilitate research, the output channels from one of the arrays, the lag array, were routed to a breakout box to allow direct access to each of the 90 channels. The outputs from the PBAs are digitized using the original system circuitry and six new analog multiplexers handling 15 detector channels each. Similar modifications to the lead array remain to be done.

The rotation stabilization synchro-to-digital converter had at least three degrees of slack in its gears. A position encoder based on two optical discs producing 60,000 pulses per rotation was included to provide a more precise angular reference signal.

The initial modification used playback from the Ampex HBR3000i into the Masscomp 530 for storage on the Optimem 1 gigabyte optical disc. Transfers to the optical disc were accomplished at 1/40th of the data recording rate using software developed by Cedar River Software [Ref 8]. It was tedious to record data and play it back hours later to see if it was acceptable. The idea of using a frame grabber board capable of accepting data at rates of up to 10 megabytes per second was advanced. The first step toward this goal was made by Engel [Ref. 2] who used a digital Input/Output to retrieve the data from the Masscomp computer and transfer it using a Direct Memory Access (DMA) controller on an IBM AT computer to a DT-2861 Frame grabber. Although this did not seem to speed up the display time significantly, it showed that the idea was feasible, and FORTRAN software programs were developed to manipulate data on the frame grabber using the FORTRAN support library functions provided by the manufacturer.

The next step toward displaying data in real time was to build an interface circuit board to handshake the data from the output of the Ampex tape recorder directly into the DT-2861 frame grabber via the external input port. This interface card was designed and built by Lentz and described by Baca [Ref. 3] who modified Engel's programs to accommodate the new frame loading and unscrambling methods.

3. Preview of the present contribution

The above method fell short of achieving the real time display objective because of inherent limitations in the speed and memory access capability of the FORTRAN unscrambling subroutine. The frame grabber memory buffers are memory mapped into the extended memory area of the 80386 computer. Because of limitations in the operating system (DOS) which restricts the 80386 computer to operation in real mode, memory addresses above 1 megabyte are not directly accessible. To access the memory buffers from real mode with a 16-bit Microsoft FORTRAN program, one had to go through a device driver by calling FORTRAN library routines provided by the frame grabber manufacturer. This process was unnecessarily slow because of limitations in the library routines provided. Blocks of frame buffer memory were transferred to FORTRAN arrays into the first megabyte of computer memory via Direct Memory Access (DMA) transfers controlled by a device driver. Since DMA transfers are limited to increments of 32 kilobytes, eight DMA transfers were necessary to download the data for a full frame. Additionally, limitations in the size of FORTRAN arrays to 64 kilobytes made the unscrambling process more difficult. Once unscrambled inside the FORTRAN arrays, the image had to be uploaded back to the frame memory buffer using once again eight DMA transfers. Using this process however, 3 degree frames were being unscrambled in approximately 2.5 seconds [Ref. 3].

Another approach would have been to use a 32 bit FORTRAN compiler and a DOS Extender however; there are a number of reasons why this was not done. First, the DT-IRIS subroutine libraries have a unique calling interface and other language specific dependencies [Ref. 12 p.4-1]. The DT-IRIS FORTRAN library specifically supports Microsoft FORTRAN 4.1 and will not compile correctly with later versions. It is not known that any available upgrade supports a 32-bit FORTRAN compiler. Second, the conversion to a 32 bit FORTRAN compiler at an expense of $1000 would require to either acquire a new FORTRAN support library at an additional cost of approximately $1500 or to program locally in FORTRAN all the board initialization procedures. This task however, is more appropriately done in assembly language since it consists mostly in programming the frame grabber board registers. Third, there were some doubts on whether or not converting the programs to 32-bit code would yield the desired real time objective and instead of investing over a thousand dollars for a 32 bit FORTRAN compiler and DOS Extender, it was decided to investigate the possibility of achieving the same objective with a one hundred dollar assembler and a shareware DOS Extender program.

The final step in displaying data in real time was accomplished during this thesis. Two important goals were achieved. First, the FORTRAN software was completely rewritten in assembly language and a DOS Extender was used to access the memory in the frame grabber directly. Second, hardware modifications to the frame grabber interface were implemented and software was written to control the start of data acquisition relative to the end of rotation (EOR) by presetting 16 bits of counter offset through an AT compatible interface board. The software involved in implementing these tasks consisted of 18 pages of 32 bit assembly language code on a 33 MHz 80386 computer. This extensive software development eliminated the need for the Masscomp 530 computer and its obsolete optical disk, and it allowed the real time display of any selected data within two seconds of its acquisition.

The ability to manipulate data in protected mode is essential to the acquisition and display of unscrambled data in real time. Data from the frame grabber can now be directly manipulated by the IBM compatible computer. Commercial display programs as well as analysis programs can be used on any AT compatible computer as opposed to the UNIX based Masscomp computer. This allows greater access and facility in data acquisition and opens the floodgate for analysis.

II. REAL TIME IMAGING

A. NATURE OF THE PROBLEM

The designation of "Real Time" for the processing and display of IRSTD output data requires some clarifications. Real time imaging in this context refers to the display of a segment of a background scene taken from the scanner rotation with a refresh rate equal to the scanner rotation speed. The above definition of real time imaging is not unique. One could conceive for instance a system in which the output data would be processed on-line and the image scrolled across the video screen at the scan rate. In such a system the complete rotation could be monitored interactively and the image stopped at a desired sector for better viewing of interesting features or targets of opportunity. This latter method however would allow very little time for signal processing or data conditioning prior to display because of the high data rates involved and would likely have to be performed almost entirely by dedicated signal processing hardware. The first approach to real time imaging was therefore adopted as a more feasible approach. Of course, as in all engineering decisions, the availability of funds and the estimated development time also weigh in the balance toward the more modest real time imaging scheme. Although slower, software based imaging has the definite advantage of being flexible and easily upgraded.

In order to qualify as real time display all image formation and processing tasks must be performed in conjunction with the data acquisition itself as opposed to during playback at a later time of recorded data.

The problem of real time imaging of the output data really consists of manipulating the data into an image form within the available time period and being able to repeat the process at every scanner rotation. A number of options are possible in determining which real time image processing tasks are to be performed between image refreshes. One task however is not optional. It consists of unscrambling the data into the image display order. As pointed out by Engel [Ref. 2] the sampling order of the detectors constitutes the core of the problem in imaging IRSTD output data. In short, the output data is out of order and must be unscrambled prior to its display in image form. The sampling order for the detectors in the two arrays starts with the second detector from each of the 12 multiplexers. This is in contradiction with the order presented by Engel [Ref. 2] and Baca [Ref. 3]. The discrepancy is introduced by a time delay between the 1/12 pulses and the sampling clock introduced by the different lengths of the lines going from the seventh floor to the scanner on the roof. In addition, detectors 75 and 90 in the lag array were found to be interchanged. The correct sampling order has been determined to be:

```
 2,17,32,47,62,77,  92,107,122,137,152,167,
 3,18,33,48,63,78,  93,108,123,138,153,168,
 4,19,34,49,64,79,  94,109,124,139,154,169,
 5,20,35,50,65,80,  95,110,125,140,155,170,
 6,21,36,51,66,81,  96,111,126,141,156,171,
 7,22,37,52,67,82,  97,112,127,142,157,172,
 8,23,38,53,68,83,  98,113,128,143,158,173,
 9,24,39,54,69,84,  99,114,129,144,159,174,
10,25,40,55,70,85,100,115,130,145,160,175,
11,26,41,56,71,86,101,116,131,146,161,176,
12,27,42,57,72,87,102,117,132,147,162,177,
13,28,43,58,73,88,103,118,133,148,163,178,
14,29,44,59,74,89,104,119,134,149,179,164,
15,30,45,60,75,90,105,120,135,150,165,180,
 1,16,31,46,61,76,  91,106,121,136,151,166.
```

Notice that 164 and 179 in the third row from the bottom are out of sequence. The sampling order of the detectors as well as the sampling circuitry assumes that all 180 detectors from the two arrays are operational. The unscrambling of the output data prior to its display is a time consuming task performed in software and can be singled out as having been the most enduring obstacle in the way of achieving real time display of IRSTD data.

The data presented at the input port of the signal processing interface arrives sequentially in the same order as the sampling order of the 180 detectors. It is then routed by an interface circuit to a frame grabber memory buffer. Each memory address on the frame grabber board corresponds to a pixel on the color monitor screen. The first buffer address corresponds to the top left corner of the screen and consecutive memory addresses are mapped from left to right, row by row, to a pixel on the screen. Each of the 16 frame buffers on the DT-2861 frame grabber board occupies 256 kilobytes of on-board memory. Thus a frame buffer can store 512 by 512 8-bit images. The frame grabber displays images by sequentially converting the 8-bit data into an analog signal which is presented to a standard RGB monitor. Each byte of memory corresponds to a pixel on the display screen. The monitor in use only displays 512 by 480 pixels; thus a frame window or active memory area on the frame grabber board must be defined with the above dimensions.

Load operations of a frame grabber memory buffer are triggered by software and occur an entire frame window at a time with a rate equal to the output rate from the tape recorder and up to 10 megabytes per second. Bytes are loaded into increasing memory locations on the frame grabber memory buffer and so a loaded frame appears scrambled. The unscrambling process is illustrated in Figure 1. Figure 1a. shows a scrambled buffer. Once unscrambled, the image consists of bands of 90 pixels high by 512 pixels wide as can be seen in Figure 1b. The 90 pixel dimension corresponds to an array of 90 detectors. The 512 pixel width corresponds to the screen pixel width. Since there are 60,000 samples per 360 degrees, 512 pixels corresponds approximately to a 3 degree field. The vertical field of view of an array is 10.5 degrees, thus each band represents a compressed or squashed portion of the scanner rotation, showing a 10.5 degree high by 3 degree wide field of view mapped into a 90 pixel high by 512 pixel wide band. Alternate unscrambled 90 degree bands belong alternatively to the two arrays. Every second band appears blank due to the fact that only one array is operational. A maximum of four complete 90 by 512 pixel bands can fit on the display screen.

In order to present the unscrambled output in a more proportioned format, a six degree image is formed by combining the data from two array bands and expanding the vertical scale by a factor of five by copying each of the 90 rows of the unscrambled array output to five rows on the display. An image with 450 by 512 pixels results as in Figure 1c. where the features are more easily recognizable. In order to fit the two bands on the 512 pixel wide screen, every second column in each band is deleted.

Better proportions are obtained however by displaying a 10.5 by 12 degree image mapped into a 450 by 512 pixels display window. Twelve degree images can be produced by selecting for display every fourth detector output. Another algorithm could conceivably display the average value of pairs of pixels or groups of four samples. However, negligible image degradation results by simply discarding the extra sample values for the following reason. Each detector is sampled 60,000 times per revolution or once every 0.1 milliradian of the rotation. Since each detector has an instantaneous field of view of 0.3 milliradian, it follows that each point in the image scene is sampled three

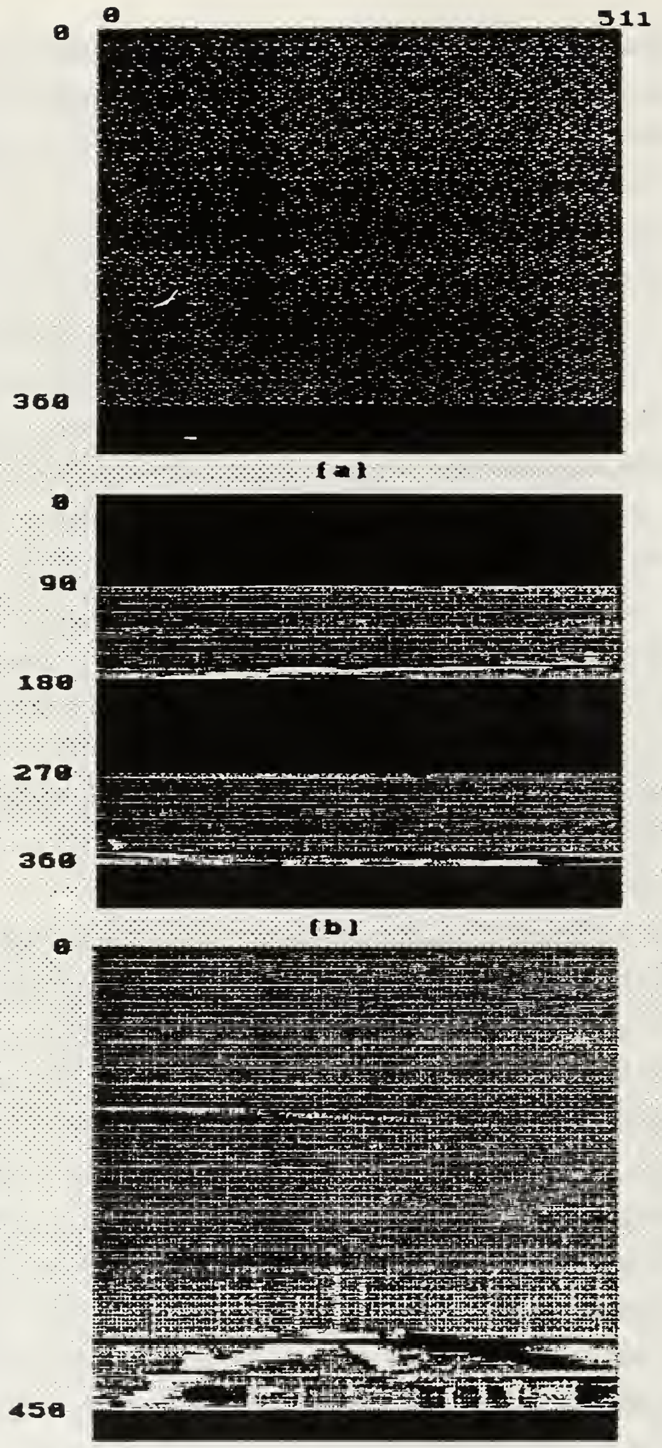


Figure 1. The Unscrambling Process
(a) 512 by 360 scrambled buffer
(b) 512 by 360 unscrambled buffer
(c) Combination of two bands in (b) plus vertical expansion by a factor of five

times during each scanner pass. As a result, discarding every second sample in the production of six degree images does not degrade the image and although some degradation occurs in the production of 12 degree images, the effect is imperceptible. On the other hand, the advantages are twofold. First, more proportioned images can be produced using a very simple algorithm in assembly language. Second, the process can easily be implemented by dividing the latching clock frequency by two or four and using a single algorithm to unscramble one band of data, 180 by 512, instead of two.

Since image unscrambling is essentially a byte manipulation operation, the idea of performing this task in assembly language to speed up the process emerged. The task seemed simple enough at first, the general scenario consisting in reading a byte from the scrambled frame buffer and directly writing it in its proper place in the next frame buffer. Conceptually, only one read and one write operation are required to unscramble an image pixel. On a 33 Mhz 80386 computer, a 256 kilobyte frame buffer could be unscrambled in approximately 32 msec assuming 2 clock cycles per operation. It quickly became apparent, however, that the task would be more complicated than originally expected. To access one byte of data in the extended memory area of the computer (memory above 1 megabyte) requires a switch over to protected mode operation. This is done with the use of a software program called a DOS extender. The issue is not trivial at all. Being a relatively recent development, DOS extenders are still very much an area of expertise reserved for software developers and professionals. Still, after long efforts at trying to optimize the FORTRAN software, the key to achieving real time display was found to be very much tied to the understanding of the protected mode operation of the 80386 computer. Because it detracts somewhat from the main subject of imaging IRSTD data and image processing of infrared data in general, a complete section on the 80386 computer protected mode operation is presented separately in Appendix A. The assembly language program itself will be described in detail in a separate section.

In summary, using an assembly language program compiled with a 386 DOS Extender, the unscrambling of six degree images from high speed data playback was achieved with a refresh time of approximately 1.5 seconds. It is to be noted that most of the time between image refreshes during high speed playback is spent waiting for the end-of-rotation signal to be detected.

A number of tasks can be performed during the extra time. In the context of imaging IRSTD data in real time, signal processing tasks can be divided into three categories:

* the image processing tasks required prior to image display,
* the desired tasks to be performed in the available time left, and
* the signal processing functions which are better left to image post processing.

In the first category we find tasks such as image unscrambling and display formatting to obtain correct image proportions for either 3, 6 or 12 degree fields. The second category comprises such operations as filling the dead detector bands or averaging damaged detector outputs with the output from adjacent detectors and adjusting the different offset values of the A/D converters which appear as bands of fifteen detectors across the screen. The third category includes all post processing tasks such as Fast Fourier Transforms, filtering and windowing etc.

B. HARDWARE DEVELOPMENTS

1. Frame Grabber Interface

The advantages of using commercial frame grabber technology to display Infrared Search and Track data have been discussed in a number of references [Refs. 2 and 3]. Data rates of up to 10 megabytes per second can be achieved, permitting real time processing and display. Frame grabber boards are computer extension cards which accept standard analog video input (TV, VCR or other) and capture or digitize 512 X 512 8-bit images at a real time rate of 30 frames per second for display on an RGB monitor. Each captured image is stored in a 256 kilobyte memory buffer on the frame grabber board. The DT-2861 frame grabber board has 4 megabytes of on board memory for a total of 16 frame buffers. Any frame buffer can be selected for display at any time. The architecture of frame grabber boards is such that images can be displayed while operations are being performed on other buffers. This is made possible through the use of multiple internal data paths. Memory buffers on the board are memory mapped into the extended memory area of the 80386 computer. The base address of the first frame buffer is selected by a jumper to be 0A00000h so as to avoid conflicts with the 8 megabytes of computer RAM in the present system. Only two addresses are fully decoded by the frame grabber board, however. Pairs of memory buffers can be accessed via the computer bus by setting the appropriate bits on the boards Output Control and Status Register (OUTCSR). The base address of the low buffer is always 0A00000h and the high buffer is 0A40000h or 256 kilobytes higher. The DT-2861 has 13 registers which control its operation. The registers are 16 bits long and occupy addresses 0250h to 025Fh. Some addresses serve for more than one register depending on values set into a third register. A more detailed description of the purpose of each register is necessary to understand how the assembly language program works; this will be given in due time.

In addition to digitizing and displaying images, frame grabber boards can perform frame-to-frame operations such as subtraction, addition, multiplication by a constant, frame averaging, graphic overlays, zooming and panning. Other useful functions include saving images to file, restoring images from files, programming color look-up tables, copying images from one buffer to another, and more. Examples of a few of these operations have been presented in Reference 6. All of these operations can be performed interactively on the DT-2861 through the use of a software program called IRIS-Tutor. Most operations can also be performed by calling FORTRAN library functions for a Microsoft FORTRAN 4.0 program. A Microsoft C Support library is also available from Data Translation [Ref. 13]. The frame grabber performs most operations in less than 1/30th of a second. More complex functions such as Fast Fourier Transforms (FFT), convolutions, windowing and histograms take longer; however, a digital signal processing (DSP) chip is available to speed these up. The present setup does not use a DSP co-processor. The DT-2861 supports false color imaging with 256 colors through the use of three look-up tables (LUT): the Input LUT, the Output LUT and the Result LUT.

The DT-2861 Frame Grabber board in use supports a number of operating modes, one of which allows high speed data transfers through an external data input port. This mode of operation bypasses the standard video input mode of the frame grabber board and needs to be interfaced to the data latching circuitry. The interface circuitry was designed and built by W.J. Lentz and was described by Baca [Ref. 4]. Figure 2 shows a diagram of this interface including the modifications added during this thesis. The external port uses a two-way asynchronous handshaking protocol. Data acquisition from the external port is enabled by software. This can be done with FORTRAN through the DT-IRIS support library or by setting with assembly language the appropriate bit in the Input Control Status Register (INCSR1) of the Frame Grabber board. Either way, the net

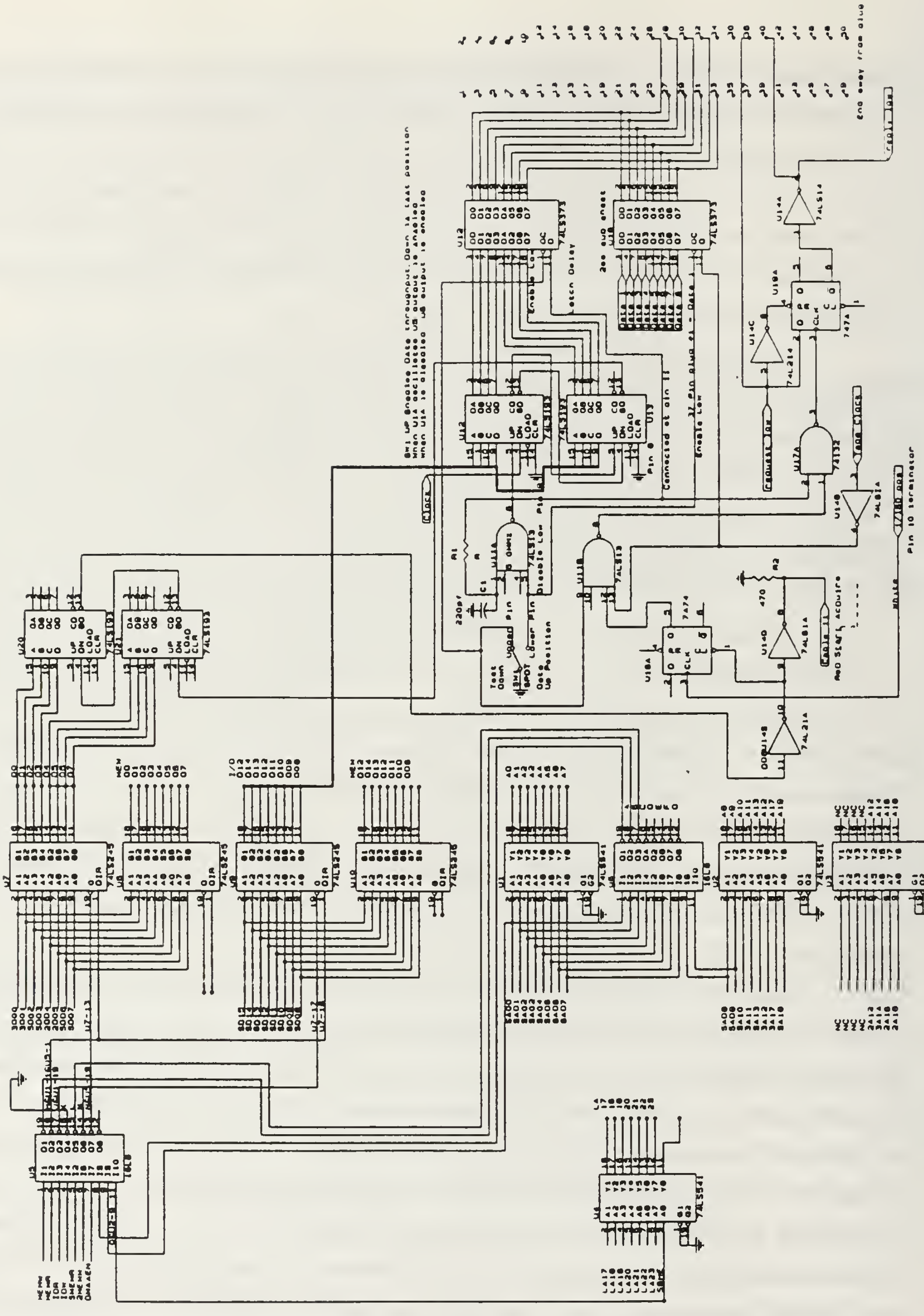


Figure 2. Frame grabber interface circuit

effect is that the BUSY signal on the board is set. This signal is used to enable a flip-flop on the interface circuit which then toggles on the first 1/180 pulse detected. The 1/180 pulse coincides with the sampling of the first detector in the array. In other words, it indicates where the top of the image is. As soon as the flip-flop toggles, handshaking of the data starts. The handshaking continues as long as BUSY is set. The Frame Grabber board automatically clears this signal when the input buffer is full. This interface had the shortcoming of loading images at random depending on where in the rotation the data happened to come from when a load was requested. IRSTD image scenes by Engel [Ref. 2] and Baca [Ref.3] have been obtained through the tedious process of playing back recorded data tapes over and over until the desired frames were obtained. The first modification brought to the interface circuitry was to trigger the load operations only at the first 1/180 pulse after the end-of-rotation signal (EOR). This signal is provided by an optical sensor which is used as a North reference for the scanner rotation. With this first change the same image was loaded and refreshed rotation after rotation although it was always pointing North. The next logical step was to delay the load operation with an offset from the EOR signal which corresponds to a desired sector in the scanner rotation. This could be done by interfacing the framegrabber circuitry with the PC AT bus so that delay counters could be set by software.

2. JDR-PR10 Prototype Board

The interface circuitry was built on a JDR-PR10 prototype card and is mounted inside the PC. The JDR-PR10 Prototype board is a PC breadboard card which provides full interfacing with the computer's AT bus. Full address decoding and chip select signals are provided by two programmable logic arrays (PLA) for port inputs and outputs. The JDR-PR10 comes with all the required components including bus and data port latches. The PLAs decode port addresses from 300h to 3FFh. These ports can be written to and read from by software at any time and can support either 8-bit or 16-bit operations. Four 4-bit counters were interfaced to port 300h in the 16-bit mode. These counters are preset by software before a frame load operation is triggered. The counters delay the beginning of the data latching from the end-of-rotation (EOR) signal. They are preset with the delay value and are decremented using the 1/180 pulses as a clock. Since there are 60000 1/180 pulses in one revolution, and the 16-bit countdown can count from 2^{16} or 65536, the counters can select the sector to be displayed with pixel accuracy.

C. SOFTWARE DEVELOPMENTS

1. FORTRAN Subroutines

Efforts were made to optimize the speed performance of the existing FORTRAN codes. Two separate programs, LOAD.for and UNSCRAMB.for [Ref. 3] performed the frame buffer load operation and the buffer unscrambling. These programs were called in sequence by a batch program. In order to speed up the process a FORTRAN Main Calling program, DISPLAY.for was written. It was discovered that attempting to unscramble a frame buffer immediately after the load operation was triggered invariably caused the program to crash. This did not occur when the two operations were performed by separate programs because of the time delay inherent to batch processing. An assembly language subroutine called BUSY.asm was written to poll for the BUSY bit (bit-5) in the Input Control Status Register 1 (INCSR1) of the frame grabber board to ensure that the load operation was completed before calling the unscrambling subroutine. In conjunction with this software addition, a slight modification to the hardware interface to the frame grabber board was made to enable the handshaking clock during load operations only if the BUSY NOT signal from the frame grabber board is clear and only after the EOR pulse has been detected. This caused each load operation to fill a frame buffer with the first 256 kilobytes of data after the EOR pulse. The unscrambling order in the original UNSCRAMB.for program was found to be incorrect. The detector digitization sequence described in references (2) and (3) to be 1, 16, 31 etc. as explained earlier is not completely correct. In actuality, the detectors are sampled starting with the third detector from each of the multiplexers. In other words the sampling order is correct but the starting point for the digitizations is not. Engel [Ref. 2] did not document explicitly this discrepancy; however, the NADDR array in the DMA.for program [Ref.2] shows the right sequence. The UNSCRAMB.for subroutine [Ref.3] was rewritten to use a look-up array instead of computing the ordering sequence and was expanded to unscramble six degree images.

A number of subroutines by Engel [Ref. 2] were integrated into the DISPLAY.for main calling program with few modifications. These routines performed the following functions:

* COLORS.for builds the color output look-up table of the frame grabber board;

* SCALE.for builds a color scale at the bottom of the display screen;

* RESP.for corrects the detector outputs based on the detector responsivity measurements by Ayers [Ref.1]. It is to be noted that detectors are numbered 1 to 90 from the bottom in Ayers' thesis and 1 to 90 from the top in Engel's Thesis [Ref.2]. The convention of numbering the detectors from the top was adopted since this corresponds to the row numbering on the display screen. However the responsivity values in the FORTRAN array XRES [Ref. 2] were not reversed accordingly. Also Engel's program contains the values for the LEAD array instead of the LAG array. At present, only the LAG array is connected.

* OFFSET.for corrects the band offset due to the different gain of the third A/D converter.

The final version of DISPLAY.for was able to display six degree pseudo-color images with a refresh time of approximately 4.5 seconds. A complete listing of the program can be found in Appendix B. The images displayed showed the first six degree sector in the rotation with reference to the North. This program was the basis for the data acquisition for the SPIE proceedings paper [Ref. 6].

It became apparent that no breakthroughs in speed would be achieved by optimizing the FORTRAN coding. For this reason, the idea of pursuing the real time display objective in FORTRAN was abandoned.

The FORTRAN library routines provided by Data Translation [Ref. 12] in support of the DT-2861 frame grabber board have been found suitable for image post processing applications. For applications where the speed of calculations is not as critical as in the real time imaging program, the ease of coding favors programming in FORTRAN rather than in Assembly Language. Engel's BEGIN.for and IMAGE.for programs [Ref. 2] are excellent examples of very useful utilities which have been modified and built upon. The FORTRAN libraries provided are quite extensive and cover a wide variety of functions. Examples of target enhancement by frame from frame subtraction are presented in reference [Ref. 6] for instance. However, for applications specific to IRST imaging such as specialized convolutionary filters and other image processing functions, the author is of the opinion that programming in C with a 32-bit C Compiler and a commercially available DOS Extender supporting both VCPI and DPMI standards for compatibility with DESQView and Windows 3.0 is the route to follow. For instance, the MTF corrections in FORTRAN as found in the IMAGE.for program take 3 to 5 minutes to complete. Speed improvements of the same order as those obtained by going to Assembly Language programs could be obtained with a 32-bit C program operating on the memory buffers of the frame grabber board directly.

2. Assembly Language Subroutines

The task of translating the FORTRAN DISPLAY program into an assembly language program was an ambitious project which spanned many months of writing an debugging. The difficulties were twofold. First, the program required a knowledge of the intricacies and the more obscure features of the 80386 computer and assembly language programming with which the author was not completely familiar at the onset of this thesis. Second, it required an understanding of the 80386 Protected Mode operation and of the role of DOS Extenders. The above topics are the subjects of many books and cannot possibly be covered in full detail in this thesis. All of the references from (14) to (23) have been used extensively. Most references are difficult, however, and for this reason, Appendix A on the 80386 Protected Mode Operation is included.

The objective of this section is to provide a coverage of the LOADUP.pm Protected Mode Assembly language program in sufficient detail that the code could be modified and recompiled by future thesis students or other potential users. For the moment, only one array of detectors is in operation. It is anticipated that slight modifications to the code will be required when the second array is made operational. Since the LOADUP.pm program interacts closely with the frame grabber hardware interface, the DT-2861 frame grabber board, the 80386 computer and the DOS 4.0 operating system, any changes to the present signal processing hardware might require a modification to the code. The code was written using MASM 5.1 but porting to the newer MASM 6.0 should be straightforward.

Before getting into the description of the LOADUP.pm program itself, some background information on the use of the PROT 386 DOS Extender and on the DT-2861 Frame Grabber register programming is necessary.

3. The PROT 386 Dos Extender

The PROT 386 DOS Extender is an Assembly Language Shareware program which first appeared in a two part installment in Dr Dobb's Journal - October and November 1990 [Refs. 19 and 20]. Although it is not a commercially available and supported software package, its publication in the Fall of 1990 coinciding with the beginning of this thesis was a stroke of good fortune which led to rapid developments toward achieving the desired goals. Of all the possible applications for which the PROT 386 DOS Extender was originally written, the one application for which it is especially well tailored is the very object of the present thesis.

The program comes equipped with its own built-in debugging aids. Unfortunately, source code debuggers such as Microsoft Codeview which is a standard utility for debugging any of Microsoft's languages including MASM 5.1 does not function with Protected Mode programs. Codeview does display the full 32-bit registers of the 386 but that is all. The PROT 386 DOS Extender has a number of BREAKPOINT macros and conditional breakpoints which when encountered, suspend the program execution, display the content of all registers and dump the stack to the screen. Breakpoint macros can be inserted anywhere in the source code and have proven to be a very efficient diagnostic tool.

The one drawback to using PROT is in the sparce amount of documentation which can be found on its use. It was written for software professionals and developers, as are most programs published in Dr Dobbs, and therefore assumes that its potential users are proficient with Assembly Language programming. In fact, one of the most fruitful sources of information on some of the macro usages is the forty page source code listing itself. This section attempts to remedy this problem by covering some of the most useful functions performed by PROT and all the functions and macros utilized in the LOADUP.pm program. It is not meant to provide a full description of the program and its capabilities but rather to provide sufficient information to the reader to be able to follow some of the functions performed in the LOADUP program and be able if need be to go in and make modifications to the source code.

Assembly language programs compiled with the PROT 386 DOS Extender are written in a file with a .PM extension. A batch file is provided to compile user programs. The PMASM.bat batch program is used as follows: PMASM filename. For example, the LOADUP.pm program is compiled by typing PMASM LOADUP at the DOS prompt. The DOS Extender and the LOADUP program use a number of INCLUDE files and libraries. It is important that the INCLUDE and LIBRARY environment variables be set properly before attempting to compile assembly language programs. Also, since a number of Microsoft languages are used on the system and different LINK.EXE files exist for each different language, care must be taken in defining the PATH environment variable so that the proper LINK utility is selected during compilation.

User programs generally have two segments: a DATA segment and a CODE segment. With PROT, the source code is written in a procedure called USER and must be inserted inside a template as shown below:

```
          PROT_DATA
          .
          .    (place user data segment here)
          .
          PROT_DATA_END
          PROT_CODE
USER        PROC
          PROT_STARTUP
          .
          .    (place user code segment here)
          .
          BACK2DOS
USER        ENDP
          PROT_CODE_END
```

The use of the PROT_CODE and PROT_DATA macros is very similar to the simplified segment directives .CODE and .DATA found in MASM 5.1 and should not be alien to assembly language programmers. The BACK2DOS macro ensures that the program executes a safe return to real mode operation and DOS. Ctrl-C and Alt-Ctrl-Del key combinations are ignored by default in PROT since rebooting in protected mode will cause the system to crash. Other than that, the user's program and data can be written as any other assembly language program using the normal instruction set of the 80386 computer. Because of the fact that some of the hardware interrupts used by the PC conflict with the interrupts the 80386 uses for error handling in protected mode, DOS and BIOS calls cannot function in that mode. PROT's strategy to circumvent this problem is to run BIOS calls as a Virtual 86 mode task. As explained in Appendix A, protected mode allows multitasking on the 386 computer. Therefore any number of Virtual 86 tasks can be started. Some DOS Extenders by comparison implement BIOS calls by returning to real mode temporarily to execute each system call. PROT emulates troublesome instructions such as CLI, STI, PUSHF, POPF, INT and IRET while in Virtual 86 mode. PROT actually runs DOS and the BIOS as a Virtual 86 task [Ref. 19].

A number of Virtual 86 calls can be found in the LOADUP program mostly for reading characters from the keyboard or displaying characters to the screen using DOS interrupt 21h functions 09h, 0Ah, 07h and 0Bh. A typical Virtual 86 task is accomplished in the following manner:

```
                PROT_DATA
message         db  "Running in 386 Protected Mode", 13, 10,"$"
                PROT_DATA_END

                PROT_CODE
USER             PROC
                PROT_STARTUP
                mov  ax, 21h
                mov PINTFRAME.VMINT, eax
                mov edx, OFFSET message
                mov ah, 9
                mov ebx, OFFSET PINTFRAME
                VM86CALL
                BACK2DOS
USER             ENDP
                PROT_CODE_END
```

The above example is a complete program by itself and can be compiled with the PMASM batch file. It uses the int 21h DOS function 09h to display the string "message" to the screen. It follows the same procedure explained in the MASM user manuals for executing the same function in real mode except that it is dressed up with macro commands such as VM86CALL and a redirection to the protected mode interrupt routine table (PINTFRAME).

PROT offers a number of routines which are very useful and easy to use. Among the routines provided the following can be found in the LOADUP program:

Routine	Purpose
CLS	Clears page 0 of the video display
OUCH	Outputs the character in register AL to page 0 of the video display
CRLF	Perform a carriage return/line feed
HEXOUT	Outputs the byte in register AL in hex
HEXOUT2	Outputs the word in AX in hex to the screen
HEXOUT4	Outputs the double word in EAX in hex

As an example of the use of the above routines, the following code fragment is given:

```
    .
    .
    .
    .
CALL32F sel_code32, CLS   ;clear the PC screen
CALL32F  sel_code32, CRLF        ;CR/LF
mov  al, 'P'                     ;output character P
CALL32F  sel_code32, OUCH        ;to the screen
    .
    .
    .
```

A number of macros are provided with PROT to generate proper negative 32-bit relative JUMPs. MASM 5.1 does not handle certain 32-bit references properly. The LOADUP program uses the JMP32S macro to perform short negative jumps in a 32-bit segment. The JUMP instructions are used in the same way as normal assembly language jumps ie. JMP32S label. Other jump instructions include JMPABS to perform absolute 16 bit jumps, JMPABS32 to perform absolute 32 bit jumps and JCC32 which uses Intel's condition codes to control 32 bit relative jumps.

3. DT-2861 Frame Grabber Board Register Programming

A set of thirteen registers control all operations and monitor the status of the DT-2861 frame grabber board [Ref. 13]. These registers are accessed using 16 bit input/output operations with the computers IN and OUT instructions. The registers are accessed through eight consecutive input/output port locations with a base address selectable with a jumper on the board. The base address is 0250h in the present setup. A list of the DT-2861 registers with their corresponding port addresses is presented in Appendix C. Note that some of the registers share the same port addresses. The actual register being addressed depends on the value of certain bits in another register. In other words, it depends on the current mode of operation of the board. The detailed function of each of the 16 bits in all of the registers is described in detail in the DT-2861 Users Manual [Ref. 13]. The DT-2861 can operate in seven different modes, only two of which are of interest to this thesis: the output LUT programming mode and the external port input mode. General programming sequences are presented in reference (13) giving each of the steps to be performed. However, actual assembly language code sequences are not given. The aim of this section is to present some of the key assembly language code sequences and to point out some of the difficulties encountered in writing the LOADUP program.

a. External Port Input Programming

This section describes how to initialize the DT-2861 frame grabber board registers for loading data from the external input port. The following steps must be performed prior to triggering a load operation:

(1) Program the input look-up table and the result look-up table to be used. This step will be described in more detail in the next section;

(2) Program the ALU bits in the INCSR1 register. For External Port Input the DT-2861 user's manual indicates use of the "F=A" mode; however, there are three possible settings of the ALU bits which give this function depending on whether the board is set to perform arithmetic operations without carry, with carry or logic operations. These functions are selected in turn by the ALUM bit (bit 3 of INCSR1) and CARRYIN bit (bit 4 of INCSR1). The only mode which has worked successfully with the LOADUP program was to set all the above bits to zero;

(3) Program the START and END registers to define the load window. A window of 512 columns by 360 rows is loaded. The 360 rows give sufficient room for four bands of 90 detectors or two bands from each array. This loads enough data to unscramble a six degree image or a twelve degree image depending on whether the hardware interface clock rate is divided by two to select every second data byte or not. The START register is set to 0000h thus selecting the top left corner of the active window to row 0 pixel 0. The END register is set to 0B67Fh, thus selecting the bottom right corner of the active window to row 360 pixel 512;

(4) Check the BUSY bit (bit 7 of INCSR1) and ensure that it is clear before proceeding. If it is set, clear the PASS bit (bit 5 of INCSR1) to stop the board operation normally at the end of the current operation. This step is required because the INCSR2 register cannot be accessed while the board is performing an operation;

(5) Set the mode bits (bits 4-6 of INCSR2) to 111 to select external port input;

(6) Select the input buffer by setting the BUFSEL bits (bits 0-3 of INCSR2). Buffer 0 is the input buffer in LOADUP so that these bits are set to 0000;

(7) Clear the WP0, WP1, WP2 and WP3 bits (bits 12-15 of INCSR2) to disable the write protect planes so that buffers can be accessed freely.

(8) Set the INSEL bits (bits 0-2 of INCSR1) to select the desired input look-up table. These bits are cleared to select Input LUT 0 in the LOADUP program.

(9) Clear the TRGEN bit (bit 6 of OUTCSR) so that the load operation can be started as soon as BUSY is set instead of being triggered by a low to high transition on EXTTRG. This step is not included in the user's manual programming sequence but is absolutely necessary.

(10) Clear the PASS bit (bit 5 of INCSR1) and set the BUSY bit (bit 7 of INCSR1) to enable the load operation.

(11) Poll the BUSY bit until the load operation is completed.

The following assembly language example goes through the above sequence except step 1 and would load a 512 by 360 window in frame buffer 0. It is a complete program which can be compiled and linked with the MASM and LINK commands. Note that the operations are all done in 16-bit arithmetic in Real Mode. Protected Mode operation is only required in the unscrambling routine.

```
                .MODEL LARGE
                .386
                .DATA
;DT-2861 Registers
INCSR1     equ     0250h
INCSR2     equ     0252h
OUTCSR     equ     0254h
STARTR     equ     025Ch
ENDR       equ     025Eh

;Operating Modes
EPORTI     equ     0070h  ;MODE=111, input buffer 0
ALU                equ     0000h
;ALU=0000,CARRYIN=0,BUSY=0,PASS=0,ALUM=0
SLINE      equ     0000h  ;start row 0, start pixel 0
ELINE      equ     0B67Fh  ;end row=360, end pixel=512

                .CODE
                PUBLIC      load
load           PROC
                 mov dx, INCSR1              ;Step (2) and (8)
                 mov ax, ALU
                 out    dx, ax
```

```
        mov  dx, STARTR         ;Step (3)
        mov  ax, SLINE
        out  dx, ax
        mov  ax, ENDR
        mov  ax, ELINE
        out  dx, ax

        mov  dx, INCSR1         ;Step (4)
        in   ax, dx
        btr  ax, 5
busy:   mov  dx, INCSR1
        in   ax, dx
        bt   ax, 7
        jc   short busy

        mov  dx, INCSR2         ;Step (5), (6) and (7)
        mov  ax, EPORTI
        out  dx, ax

        mov  dx, OUTCSR         ;Step (9)
        in   ax, dx
        btr  ax, 6
        out  dx, ax

        mov  dx, INCSR1         ;Step (10)
        mov  ax, LOADBUF
        out  dx, ax

poll:   mov  dx, INCSR1         ;Step (11)
        in   ax, dx
        bt   ax, 7
        jc   short poll

        mov  ax, 4C00h          ;Return to DOS
        int 21h
load    ENDP
        END
```

The above example program is not a protected mode program and can be compiled with MASM 5.1 as any ordinary assembly language program. It serves to illustrate how the DT-2861 Frame Grabber Board registers can be used to control the board operation. A portion of the LOADUP.pm protected mode code is identical to the above sequence.

b. Output Look-Up Table Programming

Each pixel on the DT-2861 can have a value between 0 and 255. Each pixel value acts as an index or pointer into a look-up table (LUT) which contains sets of three numbers between 0 and 255 representing RED, GREEN and BLUE intensity levels. Up to 16,777,216 different color combinations can be obtained. Eight output look-up tables (0-7) can be programmed with different selections of colors. When the board is initialized after power up of the system, the output LUTs are set to display 256 gray levels where 0 is completely black and 255 is completely white. The output color look-up table for the LOADUP program is stored in an ASCII INCLUDE file. Values in this table can be determined by using the IRIS-Tutor program and experimenting with the color palette. It is necessary to recompile the LOADUP program after changes to the LUT.inc INCLUDE file.

The output LUT register programming sequence is given below:

(1) Check if the BUSY bit is cleared as in step (4) of the external port sequence above.

(2) Save the values of the INCSR2 and OUTCSR.

(3) Set the MODE bits to 010 in the INCSR2.

(4) Select the output LUT to be programmed by setting the OSEL bits (bits 0-2 of OUTCSR). LOADUP only reprograms the output table 0 so these bits can remain clear.

(5) Select a table entry value and write it to the INDEX register.

(6) Write the corresponding RED and GREEN LUT values in the REDGRN register with the RED value (0-FFh) in the low byte and the GREEN value (0-FFh)in the high byte.

(7) Write the corresponding BLUE LUT value in the BLUE register with the value (0-FFh) in the low byte.

(8) Repeat steps 4 through 7 for all 256 LUT INDEX values.

(9) Restore the original INCSR2 and OUTCSR registers.

Steps (1) to (5) are similar to steps found in the previous section. An assembly language code fragment to execute steps (4) through (8) is presented below.

```
        .
        .
        .
        mov ecx, 0000h          ;index 0 into LUT
lut:    mov dx, OUTCSR
        mov ax, OSEL            ;Step(4),OSEL=000,BUSBUF=0,display off
        out   dx, ax

        mov  dx, INDEX          ;Step (5)
        mov   ax, cx
        out    dx, ax

        mov  edi, ecx           ;Step (6), red green and blue
        mov ebx, OFFSET red                   ;are the base addresses
        mov al, BYTE PTR ds:[ebx+edi]         ; in the LUT.inc include
        mov ebx, OFFSET green                 ;file
        mov ah, BYTE PTR ds:[ebx+edi]
        mov dx, RGLUT
        out   dx, ax

        mov ebx, OFFSET blue                  ;Step (7)
        mov al BYTE PTR ds:[ebx+edi]
        mov dx, BLUT
        out   dx, ax

        inc ecx                 ;Step (8), repeat for 256 values
        cmp ecx, 0FFh
        jle lut
        .
        .
        .
```

In the above code fragment, red, green and blue values are loaded from three arrays of 256 bytes stored in an INCLUDE data file. The INCLUDE file LUT.inc is an ASCII file which can be modified at will to give the desired pseudo-color output. The LOADUP.pm program must be recompiled however for the changes to take place.

5. The Loadup Protected Mode Assembly language Program

The mechanics of the program have already been presented in the previous three sub-sections on the PROT 386 DOS Extender, the DT-2861 external port programming and output look-up programming. This section describes the LOADUP.pm program as a whole through a set of flowcharts and covers some specific programming difficulties encountered during coding. This section should be read in conjunction with the program listing which can be found in Appendix D.

The program unscrambles a six degree frame for both the lead and lag arrays. Each row of the lag array output is then expanded by five and the resulting 512 by 450 pixel image is displayed. As mentioned earlier, this same procedure would result in a 12 degree image if every second byte of data were to be skipped during the load operation. The desired clock signal is selected by a toggle switch on the handshaking circuit board.

Figure 3 shows the overall program flowchart. The frame grabber is initialized at the beginning of the program. The board initialization is done once and consists of the the following steps:

(1) Clear the computer screen and display a welcome message;

(2) Turn on the frame grabber display;

(3) Set the display buffer to buffer 2;

(4) Stop the current board operation;

(5) Ask if programming of the output LUT is required, and

(6) Reprogram the output LUT if necessary.

During the program execution, scrambled buffers are loaded in buffer 0. Buffer 1 is used by the unscrambling process and the unscrambled, expanded image is formed back in buffer 0. Once an image is formed, it is then copied over to buffer 2. Buffer 2 is displayed continuously throughout the whole procedure and is refreshed after every scanner rotation. In the reset state the look-up tables are loaded with a monotonically increasing grey scale. The initialization step requests a choice from the user on whether or not to reprogram the output LUT for display in pseudo-color. The output LUT programming then takes place according to the steps already described in the previous section.

The load operation consists of two basic steps: the selection of the sector counter offset for the desired sector to be displayed and the programming of the external port input itself. The latter was already discussed in section C 3 (b).

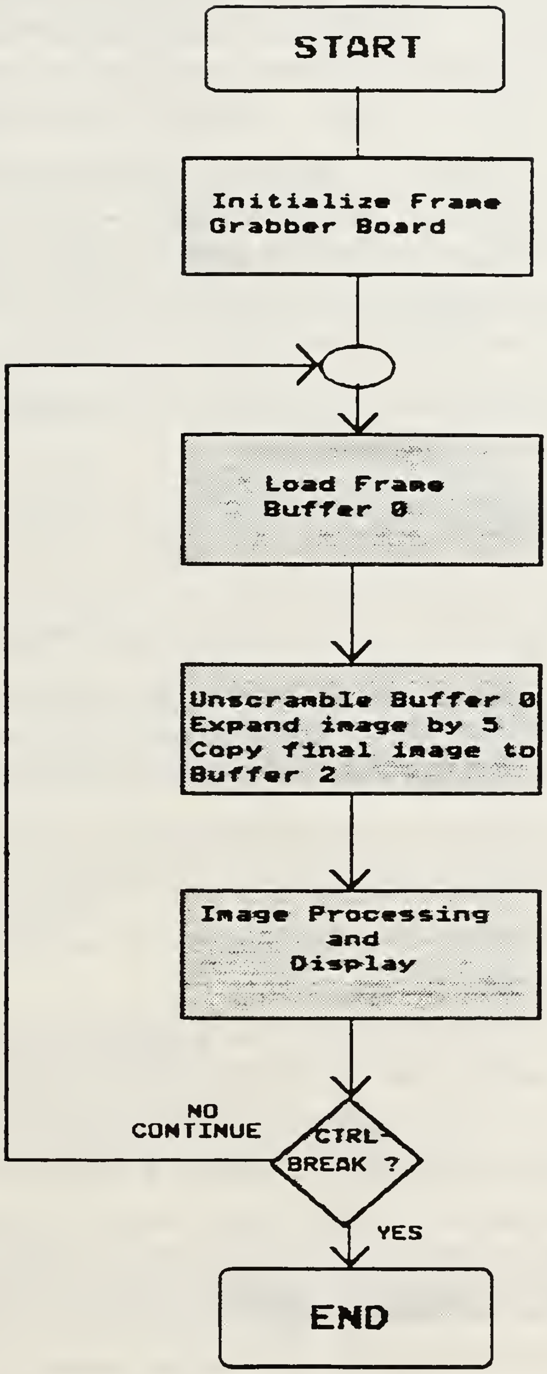


Figure 3. Flow chart of the LOADUP.pm Protected Mode Assembly Language Program

After initializing the active window in the input frame buffer, the program prompts the user for an hexadecimal offset value in the range 0000h to FFFFh. There are 60,000 sector pulses in one revolution corresponding to an hexadecimal offset of EA60h. Thus a 16 bit offset covers a little more than a complete rotation. A six degree offset is given by 0400h or 1024 sector pulses. The program does not check for the validity of the number entered; however, four digits must be entered and can be amended with the backspace key if necessary. This keyboard input is executed as a Virtual 86 call which executes the DOS int 21h function 0Ah. Once the desired offset is echoed to the screen, the string is input by pressing the enter key. The four digit ASCII string is then converted to an hexadecimal number and stored in the CX register and saved on the stack. At this point the program checks for keyboard inputs. The program recognizes the following keys from the 101 key enhanced keyboard:

*<u>Ctrl-break</u> or <u>End</u> branch to the end of the program, print a program termination string and return to DOS;

* <u>Home</u> branches to the prompt routine to enter a new sector offset;

* <u>Left and Right arrows</u> increment the sector count offset in the CX register by 0400h which corresponds to a six degree shift to the left or right. The offset value of 0400h is stored in the data segment under the label "rotate".

Once the final offset value is determined it is output to the JDR port to set the delay counters. The hexadecimal value in the CX register is converted back to ASCII characters and displayed to the computer screen and the load operation is triggered in the manner described before.

The unscrambling portion of the program reads each byte of the scrambled buffer (buffer 0) starting at address 0A00000h, looks up the correct unscrambling order from the order array in the ORDER.inc include file and forms the correct linear offset into the unscrambled buffer (buffer 1) starting at address 0A40000h. Every 180 bytes read from the scrambled buffer forms a column in the unscrambled buffer. The first three degree image is unscrambled when 512 columns are completed and fill the top part of buffer 1. The second six degree image is unscrambled the same way except that the image is stored in the bottom half of the active window in buffer 1. The expansion routine essentially merges both three degree images by ignoring every second byte in each row. The first row in the unscrambled buffer corresponds to the top detector in the lead array. The lag array starts with an offset of 512x90 or 46,080 from the start of the buffer. The second six degree image starts at an offset of 92,160 for the lead array and 138,240 for the lag array. Each row from the unscrambled lag array is copied over five rows in buffer 0. The final unscrambled and expanded image in buffer 0 occupies a 512x450 pixel window.

Image processing and display operations are then performed directly on buffer 0. An offset is added to the pixel values corresponding to the third A/D converter output to correct for the different gain in its output. The starting row for the offset operation is row 150 as specified by the "s_row" equate in the data segment. The offset operation is performed over 75 rows.

Once all image processing functions permissible during the available time are completed, the image is copied over to buffer 2 where it is displayed. This is a simple operation in principle; however, its implementation was not as straightforward as it could have been due to the ambiguity of the DT-2861 reference manual [Ref. 13]. Although frame buffers can be accessed at any time, they can only be accessed in pairs as determined by the BUSBUF bits in the Output Control Status Register (OUTCSR) of the frame grabber board. The BUSBUF bits select the two buffers accessible from the computer bus. This limitation is due to the fact that the board only decodes 18 address lines from the computer bus. The two addresses at 0A00000h and 0A40000h are selected by a jumper. The 18 address lines decoded give access to 256 Kbyte offsets from the base address of each buffer. In order to copy bytes from buffer 0 to buffer two, the following sequence must be followed:

(1) read bytes from buffer 0;
(2) change the BUSBUF bits to select buffers 2 and 3;
(3) write the bytes to buffer 2; and
(4) change the BUSBUF bits back to buffer 0 and 1.

The program then executes an absolute 16 bit jump using a macro provided by the PROT DOS Extender to the start of the next frame load operation.

III. INFRARED BACKGROUND SCENE ANALYSIS

A. BACKGROUND

Real time imaging of the IRSTD output data has provided the ability to quickly acquire, display and archive infrared scenes for the study of infrared background radiance characteristics. Interesting scene features can be selected from any part of the scanner rotation and targets can be located quickly and followed from one rotation to the next. As a result, a large library of raw video IRSTD output data files has grown rapidly.

Focussing of the optical sub-assembly in real time is yet another important technical advantage which was realized. The focus can be adjusted by monitoring a known heat source on the frame grabber display screen. It was determined that image doubling in earlier IRSTD recordings [Ref. 2] was due to misalignments in focus. A simple focus adjustment can be achieved by removing image doubling of distant antennas for instance.

This section introduces representative infrared scenes from the NPS-IRSTD and presents a number of key image processing and enhancement techniques employed.

B. IMAGING PROPERTIES OF THE NPS-IRSTD

The NPS-IRSTD is a scanning infrared sensor which is optimized for detecting and tracking targets before they come within resolution or imaging distances [Ref.8]. Consequently, the images obtained from the IRSTD differ from those of other infrared sensors specifically designed for imaging. This section explores some of the differences between IRSTD imaging and the more usual radiometric FLIR imaging.

In order to increase the sensitivity of the detectors, it is possible to increase the apparent infrared contrast by subtracting the average value of the background. This is done by coupling the detector output to a preamplifier circuit which blocks low frequency components of the detector signal. The NPS-IRSTD detector output is AC coupled with a low frequency cut-on at at approximately 100 Hz. The lack of a DC reference level in the detector output signal has direct repercussions on the type of imaging possible with the IRSTD system. Because of it, the the IRSTD system is intrinsically not radiometric. Radiometric infrared imaging systems display images in either pseudo-color or grey scale indicative of the apparent temperature in the object scene. By comparison, the IRSTD output signal fluctuates only for temperature differences and contrast in the object scene. As a result, the color gradients of displayed

IRSTD scenes are only relative. For instance, since each detector in the array is scanned horizontally across the object scene, the images displayed show color gradients only if temperature differences exist along the horizontal scan path. The drop in the apparent temperature of the sky with elevation angle is undetected by the IRSTD system. Similarly, a clear sky and a uniformly hazy sky background would appear the same if there were no temperature variations horizontally over the width of the displayed image even though the actual apparent temperatures of the two backgrounds differ. An advantage of having an AC coupled system for imaging on the other hand is that very small temperature contrasts stand out readily.

There are a few more imaging features typical of IRST systems which are directly linked with the way detectors are scanned horizontally across the object scene. Because of the fact that the IRSTD is a multi-detector independent channel system, there are bound to be differences in the responsivities of the detectors or in the gain of the preamplifiers or in a multitude of other places along the independent signal paths. The result is that the raw video display of IRST output data shows a horizontal band structure across the image. Dead or damaged detectors stand out as dark bands across the screen. Low responsivity detectors show up as strips with smaller pixel intensity offset values then the average pixel offset of the image and with smaller variations in color gradients. This effect can be corrected and smoothed out with image processing software prior to display.

Another characteristic of IRST output which is worth mentioning before proceeding with the system description is referred to as droop and undershoot response [Ref. 2]. Anomalies occur when a detector scans extended regions of high temperature (droop) or when quickly scanning from a high to low temperature region (undershoot). Undershoot appears in IRSTD images as a cold region or shadow lining the contour to the right of a hot object. Droop is another side-effect of the AC coupling scheme which is caused by a resetting of the reference level when detectors are scanned over extended regions of uniform temperature. Undershoot is a second order linear filter behavior which depends in this case on the gain of the detector preamplifiers.

Figure 4 shows a typical unprocessed twelve degree IRSTD image. It depicts the roof of Hermann Hall as well as the top of some trees to the left and clouds above. The bottom scale indicates the quantization into gray levels of the pixel intensity values between 0 and 255. The histogram of Figure 4 is shown in Figure 9a. In general, IRSTD images have a very narrow histogram with a peak at pixel values of 124 and a standard deviation of about 26. This means that using a uniform sampling [Ref. 10] or sampling with equally spaced gray levels results in loss of information in the cloud structure. The resulting picture shows most of the sky as a single gray level. Figure 5 shows the same scene after histogram equalization has been performed. The bottom scale indicates a rapid variation in the gray levels around the median at 125. Histogram equalization will be discussed shortly. For the moment, Figure 5 can be used to illustrate some of the typical features of IRSTD images. As a convention [Ref. 2] detectors are numbered from top to bottom. The black row across the bottom half of the image corresponds to a dead detector (detector 76). Low response detectors appear as horizontal lines across the image. The low response or dead detectors in the lag array are detectors: 10, 13, 14, 15, 16, 30, 41, 45, 60, 61, 76, 82, 89 and 90, in agreement with the findings in the NPS-Boeing experiment of 1989 [Ref. 7]. This fact only became evident after discovering the correct starting point in the unscrambling sequence as explained earlier. The findings in the NPS-Boeing experiment indicate that detectors 90 and 75 are switched. This is not correct. The effect of sampling the detector array by starting with the first detector of multiplexer 2 is to displace row 90 to row 75. Row 75 in turn was displaced to row 60; row 60 to row 45; row 45 to row 30 etc.

Another troublesome feature becomes more evident with the contrast enhanced image in Figure 5. The bottom 30 rows show a regular vertical noise pattern which is introduced presumably by the last two multiplexers. This source of the noise pattern has not been traced back and identified by the time of writing of this thesis.

C. NPS-IRSTD IMAGE PROCESSING

One of the most frustrating realizations of this thesis was the fact that the images displayed on the 256 color frame grabber monitor could not be reproduced faithfully in either color or gray scale in the final paper. There are many reasons why this is so. First, the images must be processed using software packages which support 512 by 512 pixel images in 256 colors which is not a standard VGA graphic mode. Secondly, a high quality printer is needed to print in 256 colors or gray levels and printer drivers must be available to support the image processing software. Thirdly, the pseudo-color IRSTD images must be mapped to gray scale images for thesis or publication purposes. Although the average human eye can easily distinguish thousands of color shades and intensities, it can only detect a few dozen shades of gray [Ref. 10]. By nature, infrared background scenes feature very small temperature differences which translate to very small contrast

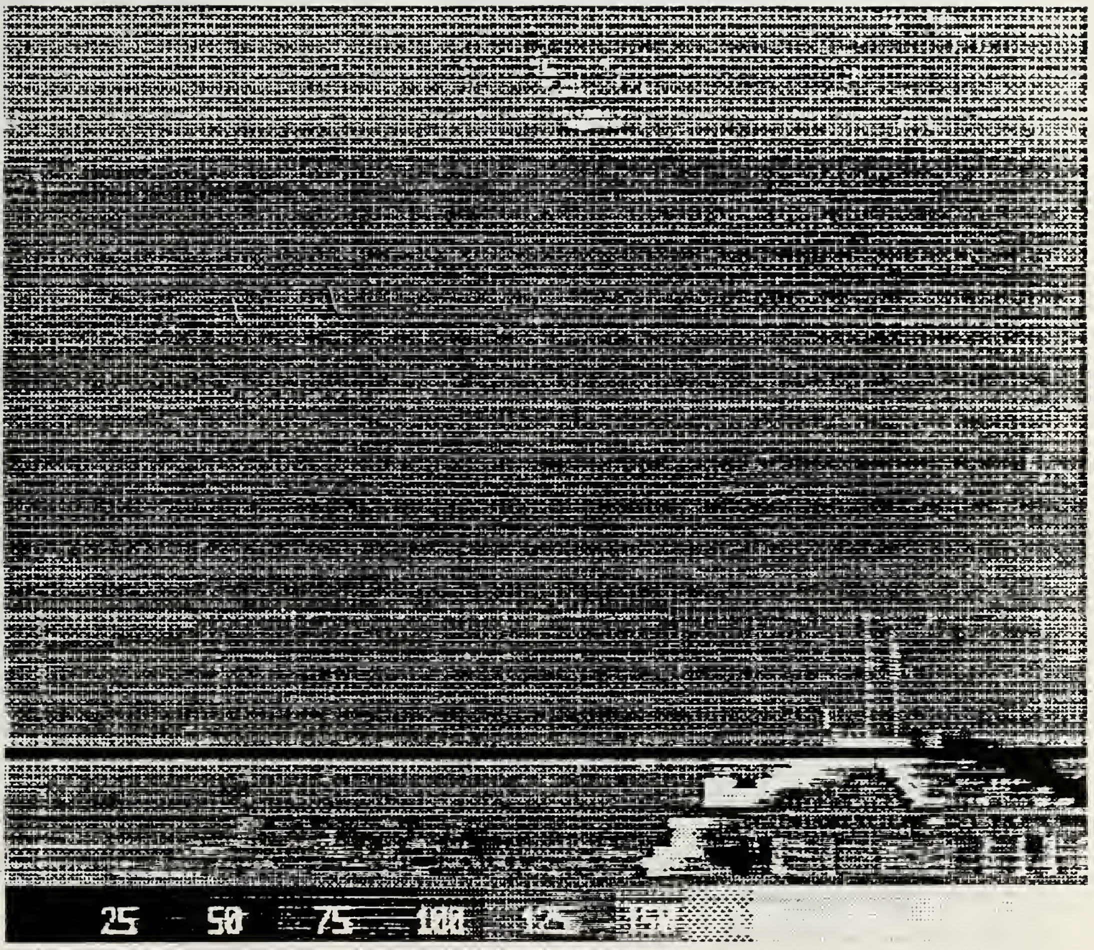


Figure 4. Unprocessed 12 degree Image of Hermann Hall

LAB ARRAY

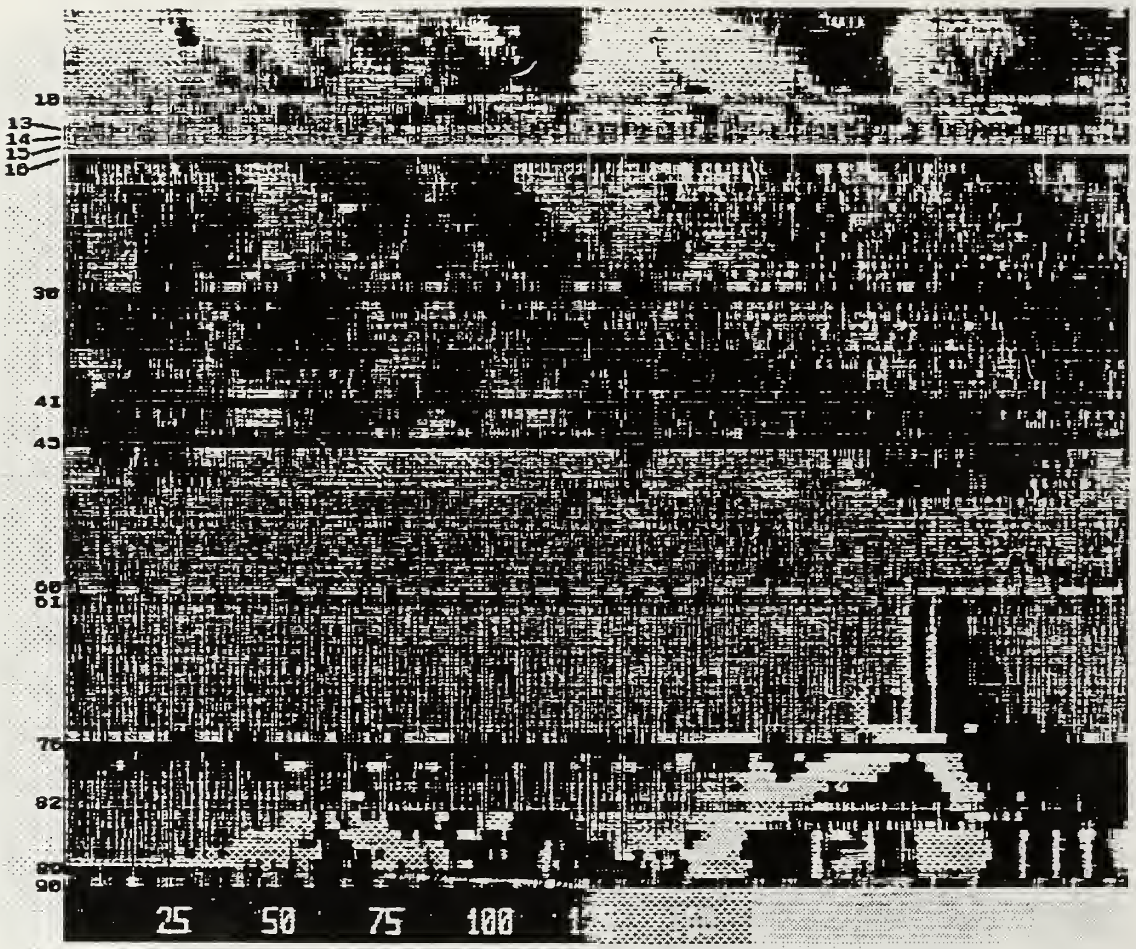


Figure 5. Histogram Enhanced, 12 degree Image Hermann Hall. Numbers on the left indicate the dead or low responsivity detectors.

differences in the image. The mapping of small variations in temperature to contrasting pseudo-colors is a powerful infrared imaging technique. There is no restriction on which color to choose for a given temperature. The problem with gray scales, however, is that they must follow rules: cold objects are darker (black) than hot objects (white) or vice versa. Finally, even if a high quality print of an image is produced, further degradation is introduced by the reproduction through photocopying of the original thesis pages.

Frame-grabbed image files are raw video data files meaning that each pixel in the image is described by an 8-bit binary value which permits a range of 256 colors or gray levels. The 8-bit binary values give only pixel intensity information; color information is not stored in this file. Instead, the colors displayed on the DT-2861 Frame Grabber monitor depend on the state of the Output LUT. The 512 by 512 pixel image is saved in a file 256 KBytes long with a 512 byte header. The process of printing an image is a three step process. First, the image file must be read by a program which recognizes the 8-bit image data format and either skips over or decodes the file header. This program must then convert the image data to a standard VGA display mode (640 by 480 pixels in 16 colors or 320 by 200 in 256 colors) and display it on the computer VGA screen. The reason for this is that the computer VGA display screen is the only one which can be accessed by commercial image processing software. Finally, a terminate and stay-resident (TSR) program must run in the background which permits image capture for printing or saving to one of many standard image file formats (TIFF, GIF, PCX etc.). The advantage of converting image files to PCX or TIFF format is that they can easily be read by the Publisher's Paintbrush or Windows Paintbrush programs and imported into either the Word Perfect 5.1 or Word for Windows word processors. A variety of programs also permit the conversion between the various image file formats. This three step process introduces compatibility problems between the various image processing software programs. A set of three programs must be found which all support the same VGA graphics card. Although SuperVGA graphics cards exist which can support 1024 by 768 pixels in 256 colors, very few programs support this mode of operation. To complicate the problem further, each card manufacturer implements SuperVGA graphics differently and although a standard (VESA) was developed by card manufacturers, a large number of software vendors still have to upgrade their products to support this standard. Therefore, the software packages used for image processing must all support a specific card if SuperVGA is to be used. This reduces the portability of the final image files produced. In other words, contrast-enhanced images in 256 colors produced with a

SuperVGA graphics mode and saved in a PCX file format are extremely unlikely to work on a different SuperVGA card even if both cards are supported by the software package generating the PCX file. Even if a high quality 256 color SuperVGA image were produced, a major difficulty would still remain in printing it with the same resolution and color or gray scale contrast. This problem is twofold: first a printer must be found which produces high resolution 256 color or grey scale output and secondly a software program or printer driver for Paintbrush must be found to support this printer. High quality color laser printers are extremely costly and there are no guarantees that such a printer would be supported by all or part of the software used. After many months of trying to produce reasonably good printouts of IRSTD images for this thesis and other papers it has become the author's opinion that the approach to follow is to acquire the hardware which will do the job and develop custom-made software for it.

The images presented in the rest of this chapter were produced using three programs: VGAIPS, Pizazz Plus and Publisher's Paintbrush. They were produced in 16 colors or grey levels and printed on a HP PaintJet printer. The final result is a much degraded image which does not do justice to the effort involved in displaying the original image in real time.

1. Image Processing Software

a. VGAIPS

VGAIPS is an image display program which was designed specifically for infrared imagery. It is used to display 8-bit binary image files from the LANDSAT and EOSAT satellites, GL Flir, GL SAIRS and IRAMMP imaging flirs [Ref. 25]. It can read any 8-bit image file such as TIFF files or frame-grabbed IRSTD files provided that the number of header bytes to skip over is known. VGAIPS supports two standard VGA modes: 640 by 480 in 16 colors (mode 12) or 320 by 200 in 256 colors (mode 13). The former is used. It also supports other modes on specific graphic cards. The program is completely menu driven. It can display images in either pseudo-color or gray scale and features a number of useful functions such as histogram equalization and direct LUT editing. Histogram plots can be displayed.

b. Pizazz Plus

VGAIPS must be used in conjunction with a memory resident utility to capture the screen. Capture programs typically take over the Print Screen key on the keyboard and allow any displayed image to be either printed directly or saved to disk with a requested file format. Screen capture programs of various kinds exist including:

Grafplus, Graflasr, VGACAP, Pizazz Plus and HiJack to name a few. All of the above support the VGA mode 12 used by VGAIPS. The last two support some SuperVGA cards and a number of printers including the HP Paintjet. These programs run in the background and necessitate the use of another program to display the image to the screen. Unfortunately VGAIPS does not display images using any SuperVGA mode. Pizazz Plus was used mostly because because it supports both the HP Paintjet and LaserJet Printers and can save images from any graphics mode (VGA or Super VGA) to PCX format.

c. Publisher's Paintbrush

This program was used to read the PCX files produced by Pizazz Plus and do the final editing on the images before printing. Publisher's Paintbrush and Windows Paintbrush support an extensive list of printer drivers among which is an excellent HP Paintjet driver. Paintbrush was used primarily to annotate the images and figures throughout the thesis. It was also used to convert color images to gray levels and to produce composite images using its cut and paste utilities.

2. Image Enhancement Techniques

As already pointed out, the images in this section are degraded compared with the images displayed on the frame grabber video display. Nevertheless, important image enhancement techniques and procedures can be demonstrated. The reader must bear in mind that the actual image displayed on the screen is much better than can be reproduced on paper with the current methods and that the contrast enhancements performed on the same images also turn out better on the video display than they do here. Figure 6 is an example of the kind of degradation inflicted on the image scenes by the printing process. Figure 6a depicts a video camera recorded image of Hermann Hall captured in 256 gray levels by the frame grabber board. This image is then saved to file, displayed by VGAIPS in sixteen colors in VGA mode 12 and converted to a PCX file by Pizazz Plus. The resulting sixteen color PCX image file was imported into Paintbrush along with Figure 6b which shows a twelve degree IRSTD image scene which was obtained using the same method. The cloud structure in the top image clearly shows the sampling effect [Ref. 10] introduced by going from 256 gray levels to sixteen gray levels. Instead of smooth or gradual changes from one gray level to the next, we observe abrupt or coarse changes which are further aggravated by the choice of the hatching pattern selected by the printer driver to represent a given gray level. Although they can be observed on the video display and in the infrared image, the antennas on the roof of Hermann Hall do not appear in Figure 6a due to the degradation just described.

The following two sub-sections will describe the techniques employed to enhance the contrast of IRSTD images. The techniques fall in two categories: those methods which can be performed directly on the frame grabber video display and the methods which employ VGAIPS or potentially other image processing software. This section describes only the practical approaches and the procedures for enhancing the images; it does not provide a full theoretical coverage of all the methods used. A complete review of digital image processing would extend beyond the scope of this thesis; however, an excellent starting point for this is reference (10).

a. Image Enhancement on the Frame Grabber Board

The DT-2861 frame grabber board is supported by a FORTRAN support library which includes a number of convolution filters which can be used for image enhancement. The images shown in this section were obtained by applying the desired filters to an image frame through the use of the IRIS-Tutor interactive frame grabber board control program. The convolutionary filters used by IRIS-Tutor are 3 by 3 pixel masks. A complete frame is convolved in three minutes on a PC/XT machine with a co-processor according to the frame grabber documentation [Ref. 11]. On the 80386 33 Mhz computer, the operation takes approximately 20 seconds to complete.

Figure 7 shows the result of applying four different masks to the scene in Figure 4. Figure 7a shows the result of a Lowpass filter. The effect of this filter is to remove the high spatial frequency components in the image. The net effect is to smooth out some of the noise structure in the image. At the same time, it widens the histogram slightly so that the clouds start to appear more contrasted. Figure 7b results from applying a Highpass filter. This filter enhances all the high spatial frequency components in the image. Although it brings out the noise in the sky portion of the image, it also enhances well the contrast of the trees and Hermann Hall. Figure 7c is the output from Laplacian edge enhancement and Figure 7d is the result of Vertical edge detection. Both edge enhancement filters clearly show the vertical noise structure which appears in the bottom third of the image as discussed earlier.

Figure 8 shows the effect of contrast enhancement through successive applications of the Offset and Multiply operations on the frame grabber board. Figure 8a is the original IRSTD image. Figure 8b was obtained by offsetting all pixel values in the original image by -62 and multiplying the result by two. Figures 8c and 8d were obtained by repeating this operation. The effect of the Offset and Multiply operation is essentially to widen the histogram. Figure 9 shows the corresponding histogram of all four images.

The mean in the original image is about 124; thus an offset of half that value or -62 was selected. By multiplying all the pixel values in the resulting image by two, the mean is reestablished, but the standard deviation is increased. This method has proven to be quite powerful since both Offset and Multiply operations on the frame grabber board take less than 1/30th of a second to perform on a full frame.

b. Image Enhancement using VGAIPS

VGAIPS provides a number of useful functions in its pull-down menus to perform histogram manipulations. The contrast of an image can be enhanced through histogram equalization or direct histogram specification or simply by editing the LUT directly. LUTs can be saved to files and used on different images if necessary. Figure 10 shows two examples of histogram modification techniques. The original image appears in Figure 10a. Figure 10b is the result of histogram equalization. The result of this nonuniform sampling approach is to increase the contrast in the cloud structure. This method works particularly well in the sky area which is relatively uniform to begin with. However, the method is not as effective for enhancing ground features or more dense areas in the image [Ref. 10]. The contrast of both trees and Hermann Hall is too high and appears as if only two or three gray levels were used. It was found that direct LUT editing yielded the best results in enhancing ground features, although most of the cloud structure is lost. Figure 10c is an example of histogram specification through direct editing of the LUT. The LUT used can be found in Figure 11. Pixel intensity values from 0 to 255 are mapped into sixteen gray levels ranging from 0 (black) to 15 (white). The gray levels are then mapped into 16 preselected colors by the software program.

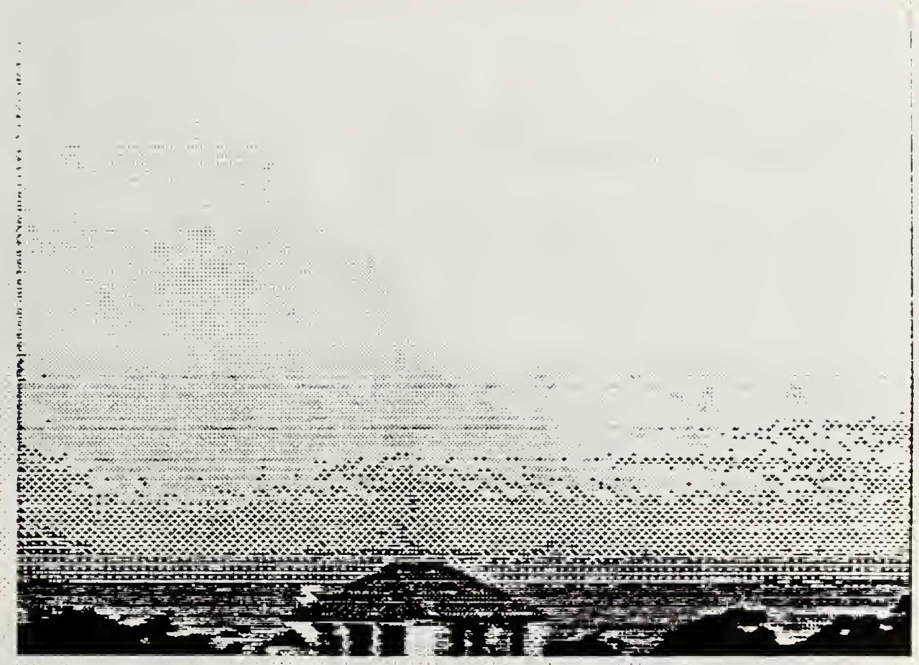
Figure 12 depicts the first 72 degrees of the scanner rotation. It is a composite of six histogram-equalized twelve degree images, put together using the cut and paste utilities in Paintbrush. The hexadecimal offset to the beginning of each twelve degree segment is annotated at the bottom of the figure. Figures 13 through 17 show a complete scanner rotation. Each twelve degree image was enhanced by direct histogram specification using the LUT in Figure 11 to enhance ground features. Once again, the hexadecimal offset of the beginning of each 12 degree segment is indicated and should be used as a reference to locate a position in the scanner rotation using the LOADUP.pm display program. The inclusion of Figures 13 through 17 in this thesis is intended to provide documentation on a complete scanner rotation to facilitate the location and identification of background scene features during future data recordings. The real time imaging program displays a six degree portion of the complete rotation. Viewing the data through such a small window makes scene features difficult to locate and recognize. A complete rotation already documented is paramount to the detection and tracking of an aircraft in real time. With a complete rotation already documented, an observer positioned on the roof could provide the operator on the second floor the accurate angular positions of aircrafts approching the airport or taking off or of boats in the harbour. Easily recognizable features have been labeled at the top of the figures and can also be used as reference points.

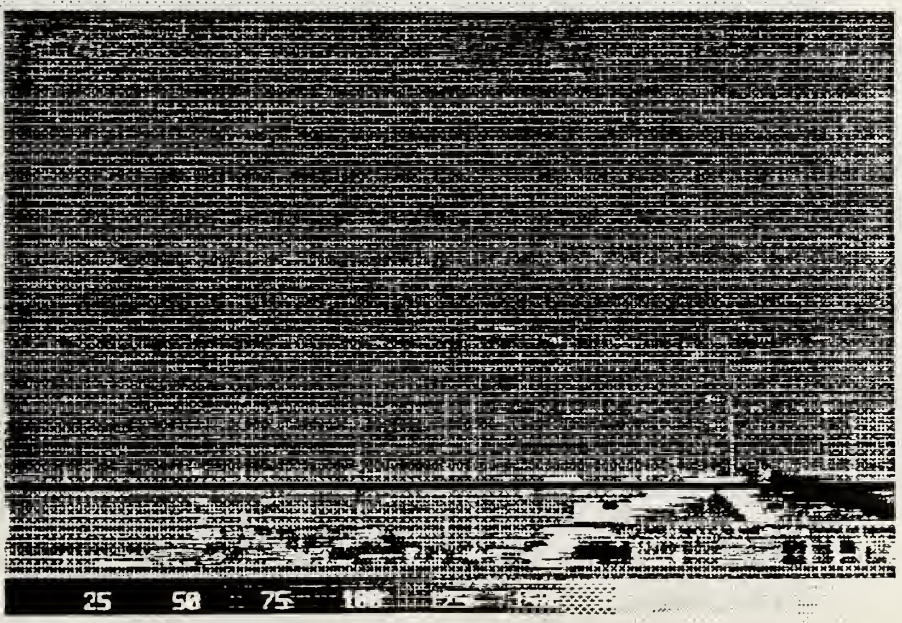
Figure 18 is another composite picture put together using Paintbrush. It shows a US Air PropJet passenger plane landing at the Monterey Airport. The full width of the picture is approximately 15 degrees. The airplane flew slightly to the left of the scanner position and followed a straight line path away from the scanner. It was followed in real time for over eight scanner rotations. The image was reconstituted by capturing each consecutive scanner pass during playback of the recorded tape. The individual airplane images were then pasted together to produce this composite picture. The blurred appearance of the airplane in position 1 and 2 is possibly due to the fact that the airplane is closer to the scanner and is in motion as the IRSTD scans across it. During the remainder of the landing the airplane is seen approximately from the tail aspect as it moves away from the scanner. It appears as a pair of white, or high temperature, spots which can possibly be recognized as the two propjet engines. Undershoot to the right of the airplane is evident and is an artifact of the AC coupling on the analog bandpass filters.

c. Summary

This section introduced a number of methods and developed some of the the tools which will be useful for the analysis and study of infrared background scenes using the NPS-IRSTD. Some of the key imaging properties of the scanner were pointed out and explained. Basic image enhancement techniques using commercial image processing software were described and typical infrared scenes were presented including a banner showing a complete 360 degree rotation.

A composite picture made of eight consecutive scanner rotations showing an aircraft landing at the Monterey airport was presented as an example of the newly gained ability to selectively acquire and display IRSTD data using the LOADUP real time imaging program.


(a)



(b)

Figure 6. Comparison between a VCR recorded image (a) and an unprocessed IRSTD image (b). Both images in 16 gray levels.

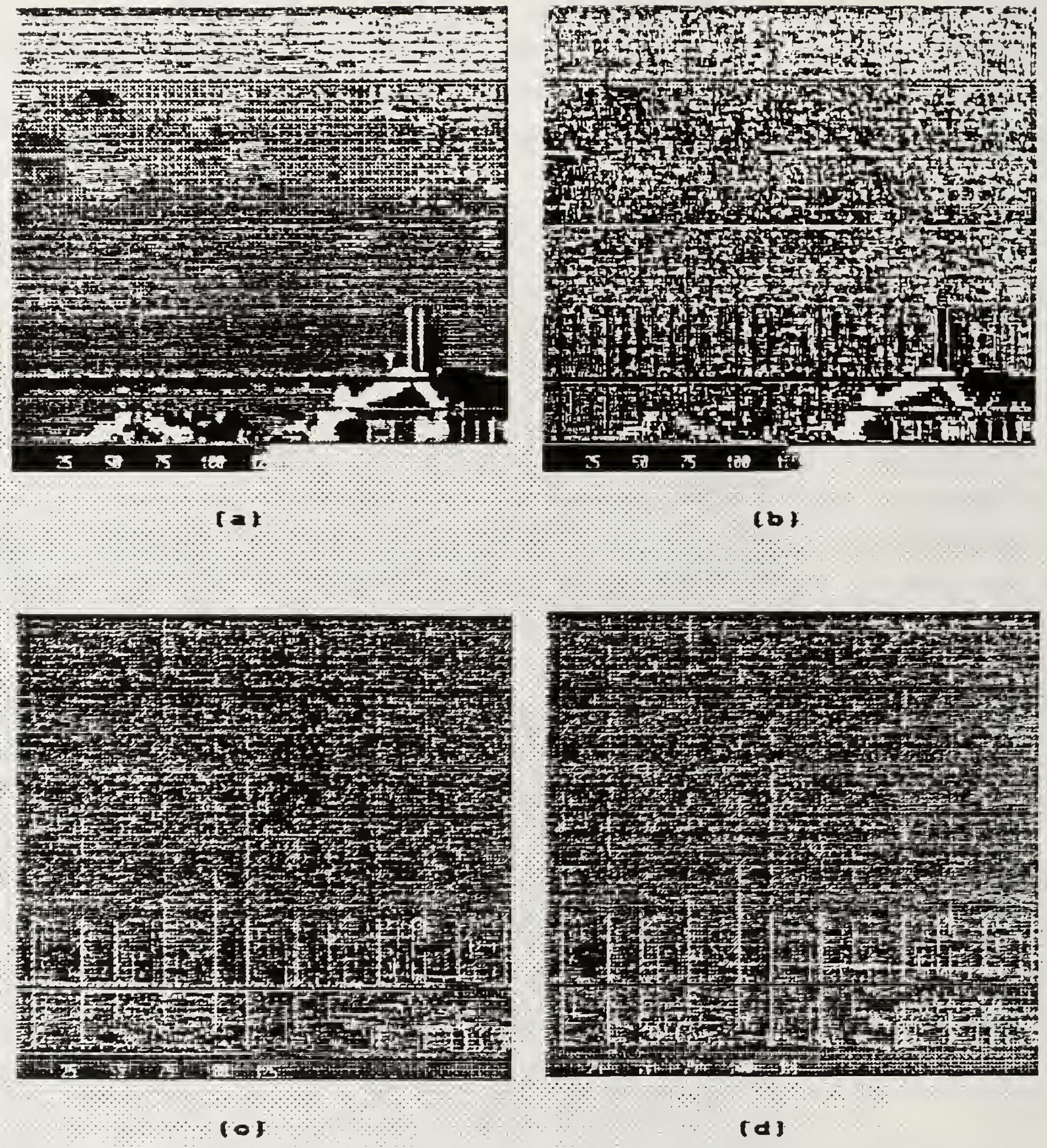


Figure 7. Examples of image enhancement techniques on the frame grabber board. (a) Lowpass filtering. (b) Highpass filtering. (c) Laplacian edge enhancement. (d) Vertical edge detection.

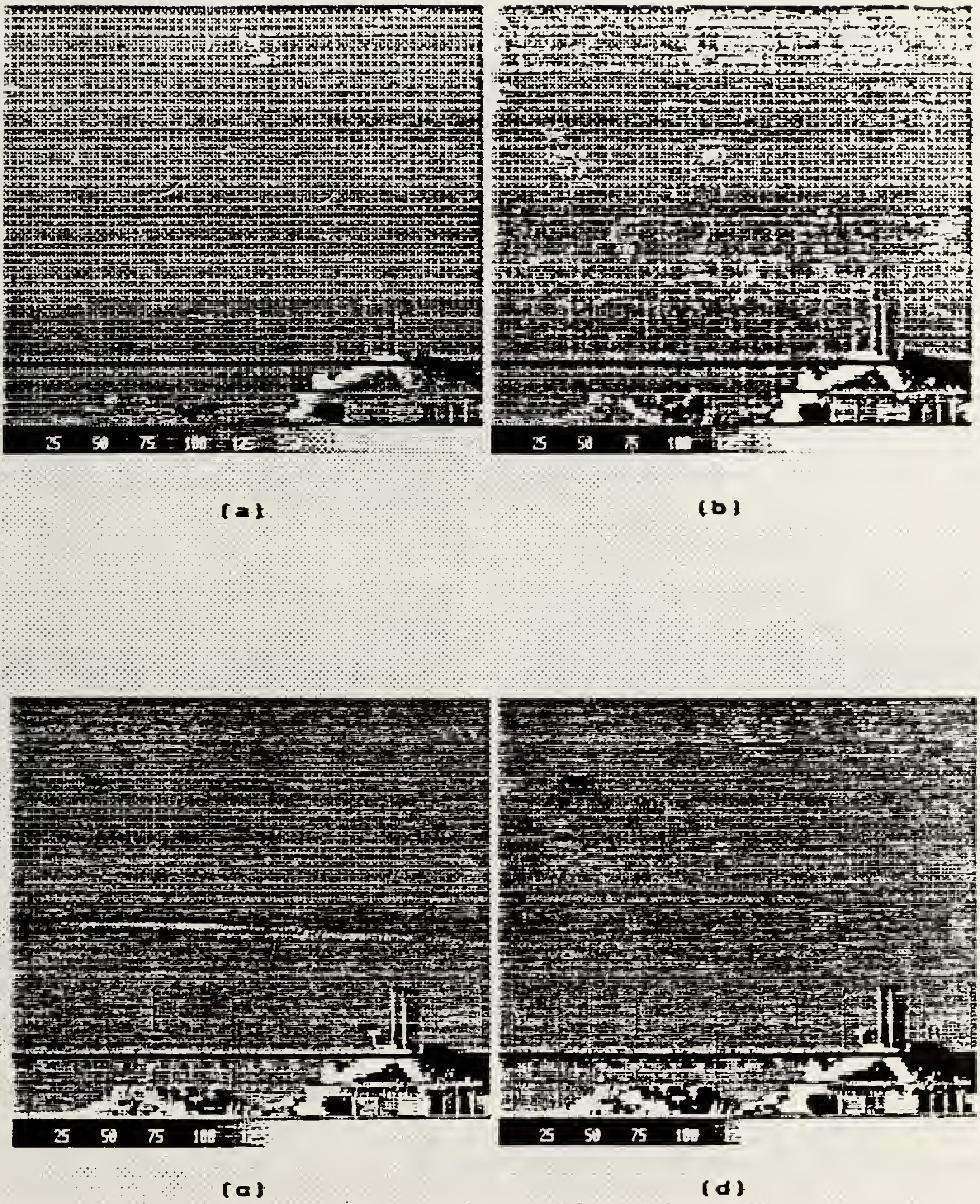
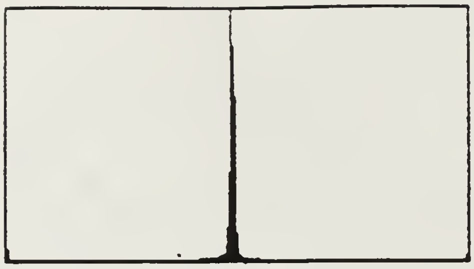
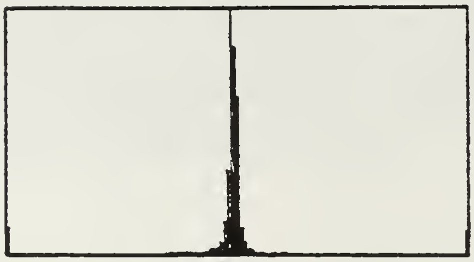


Figure 8. Image enhancement using offset and multiply operations on the frame grabber board. (a) original image. (b) offset by -62 then multiply by 2. (c) Same operation applied to (b). (d) Operation applied to (c)



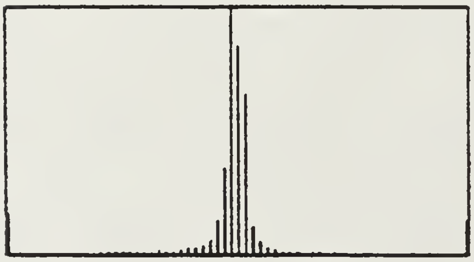
```
Peak value =  64710  at  124
Minimum value =  0
Maximum value =  255
Histogram mean = 123.946
Histogram median =  125
Standard deviation = 26.456
```

(a)



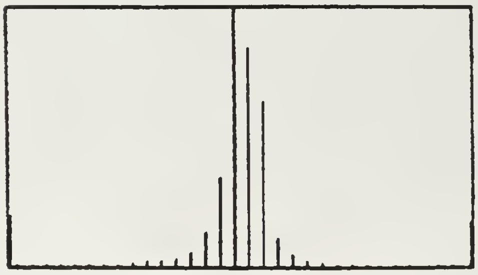
```
Peak value =  64710  at  124
Minimum value =  0
Maximum value =  255
Histogram mean = 124.373
Histogram median =  126
Standard deviation = 33.782
```

(b)



```
Peak value =  64710  at  124
Minimum value =  0
Maximum value =  255
Histogram mean = 125.836
Histogram median =  120
Standard deviation = 38.943
```

(c)



```
Peak value =  64710  at  124
Minimum value =  0
Maximum value =  255
Histogram mean = 126.531
Histogram median =  132
Standard deviation = 44.48
```

(d)

Figure 9. Histograms for the pictures in figure 8. The effect of the offset and multiply operation is to widen the histogram.

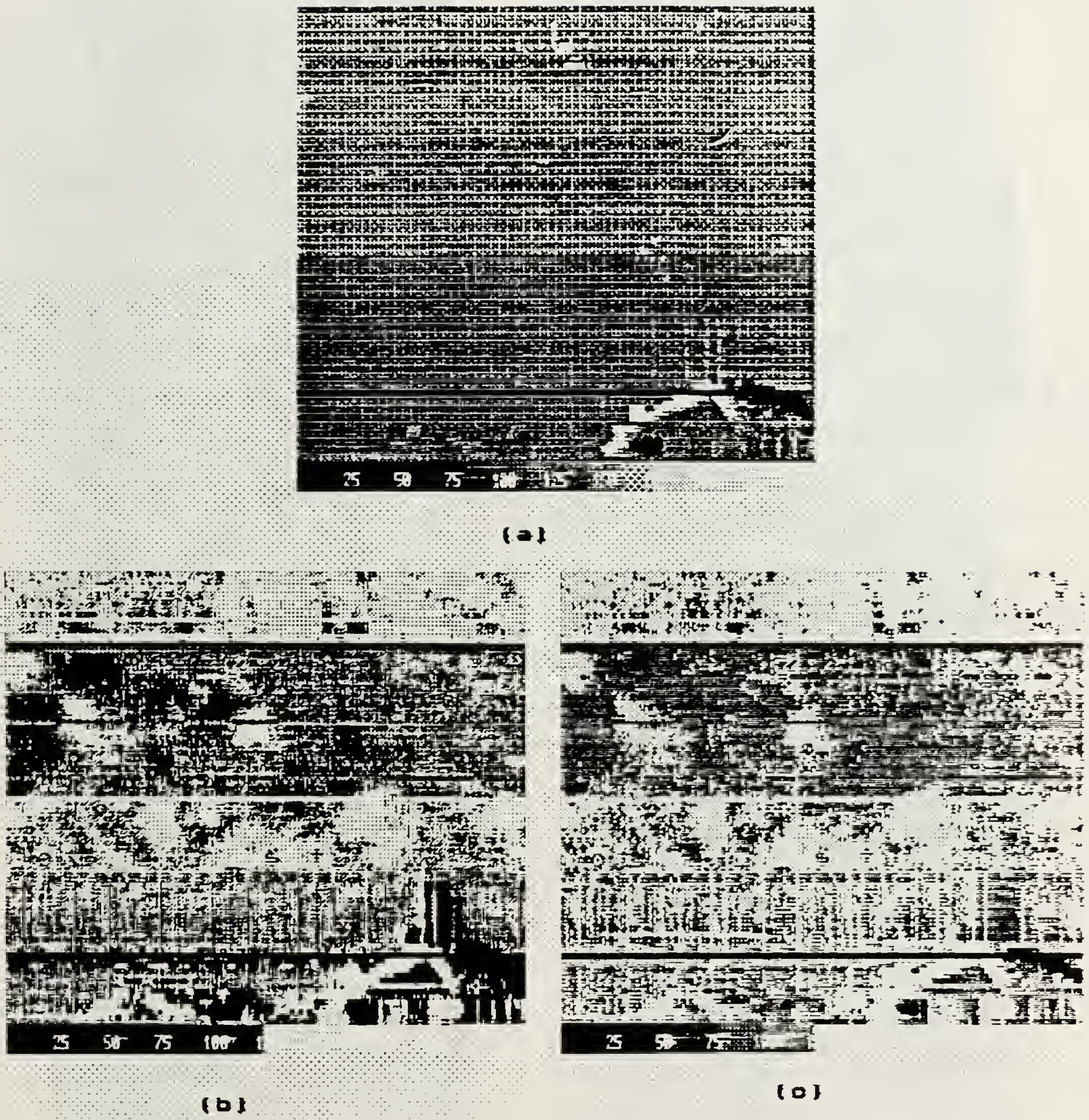
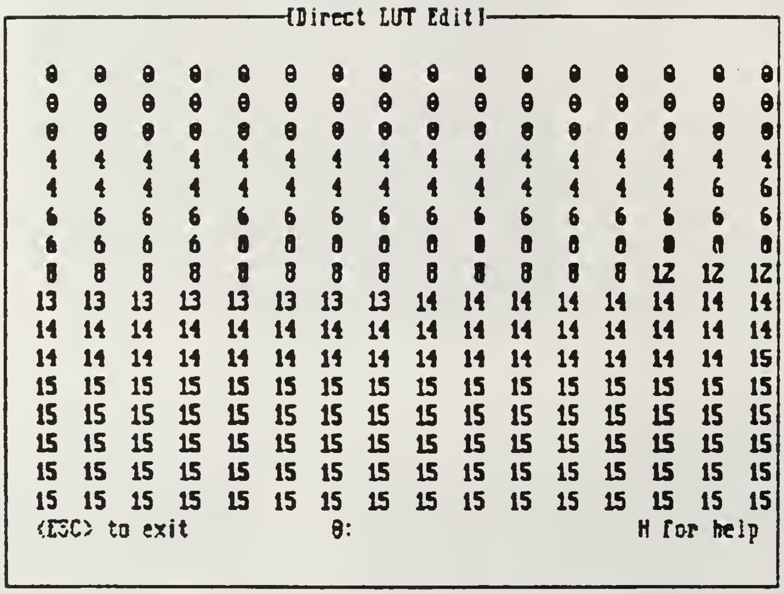


Figure 10. Examples of image enhancement using histogram manipulation techniques. (a) unprocessed image. (b) Histogram equalization. (c) Direct Look-up table specification.



```
{Direct LUT Edit}

  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
  4   4   4   4   4   4   4   4   4   4   4   4   4   4   6   6
  6   6   6   6   6   6   6   6   6   6   6   6   6   6   6   6
  6   6   6   6   8   8   8   8   8   8   8   8   8   8   8   8
  8   8   8   8   8   8   8   8   8   8   8   8   8  12  12  12
 13  13  13  13  13  13  13  13  14  14  14  14  14  14  14  14
 14  14  14  14  14  14  14  14  14  14  14  14  14  14  14  14
 14  14  14  14  14  14  14  14  14  14  14  14  14  14  14  15
 15  15  15  15  15  15  15  15  15  15  15  15  15  15  15  15
 15  15  15  15  15  15  15  15  15  15  15  15  15  15  15  15
 15  15  15  15  15  15  15  15  15  15  15  15  15  15  15  15
 15  15  15  15  15  15  15  15  15  15  15  15  15  15  15  15
 15  15  15  15  15  15  15  15  15  15  15  15  15  15  15  15
 <ESC> to exit            0:                        H for help
```

Figure 11. VGAIPS screen showing the look-up table used for figure 10 (c)

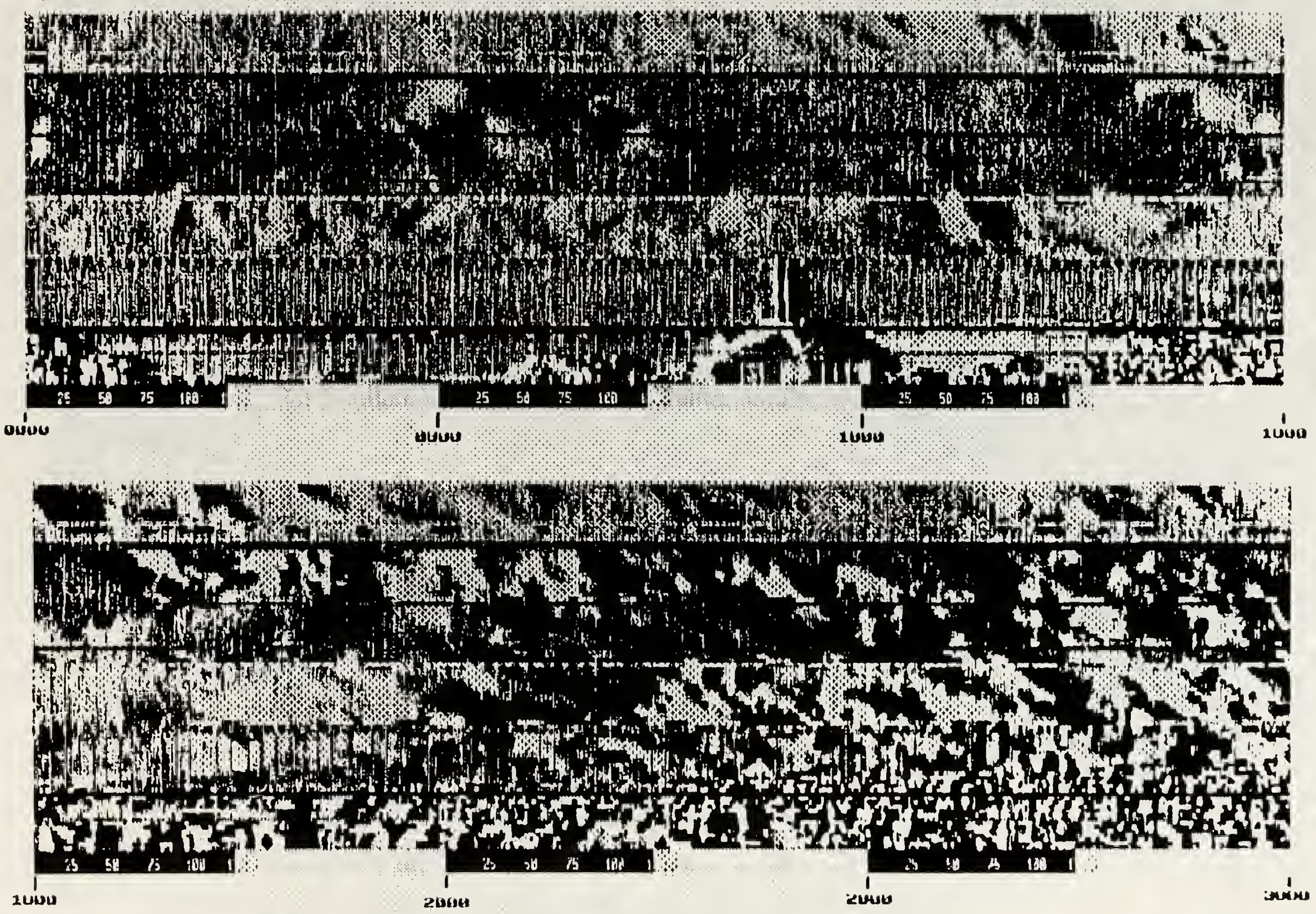


Figure 12. First 72 degrees of the scanner rotation. Histogram equalization applied to enhance cloud formations, Hexadecimal offset of the beginning of of each 12 degree frame is indicated.

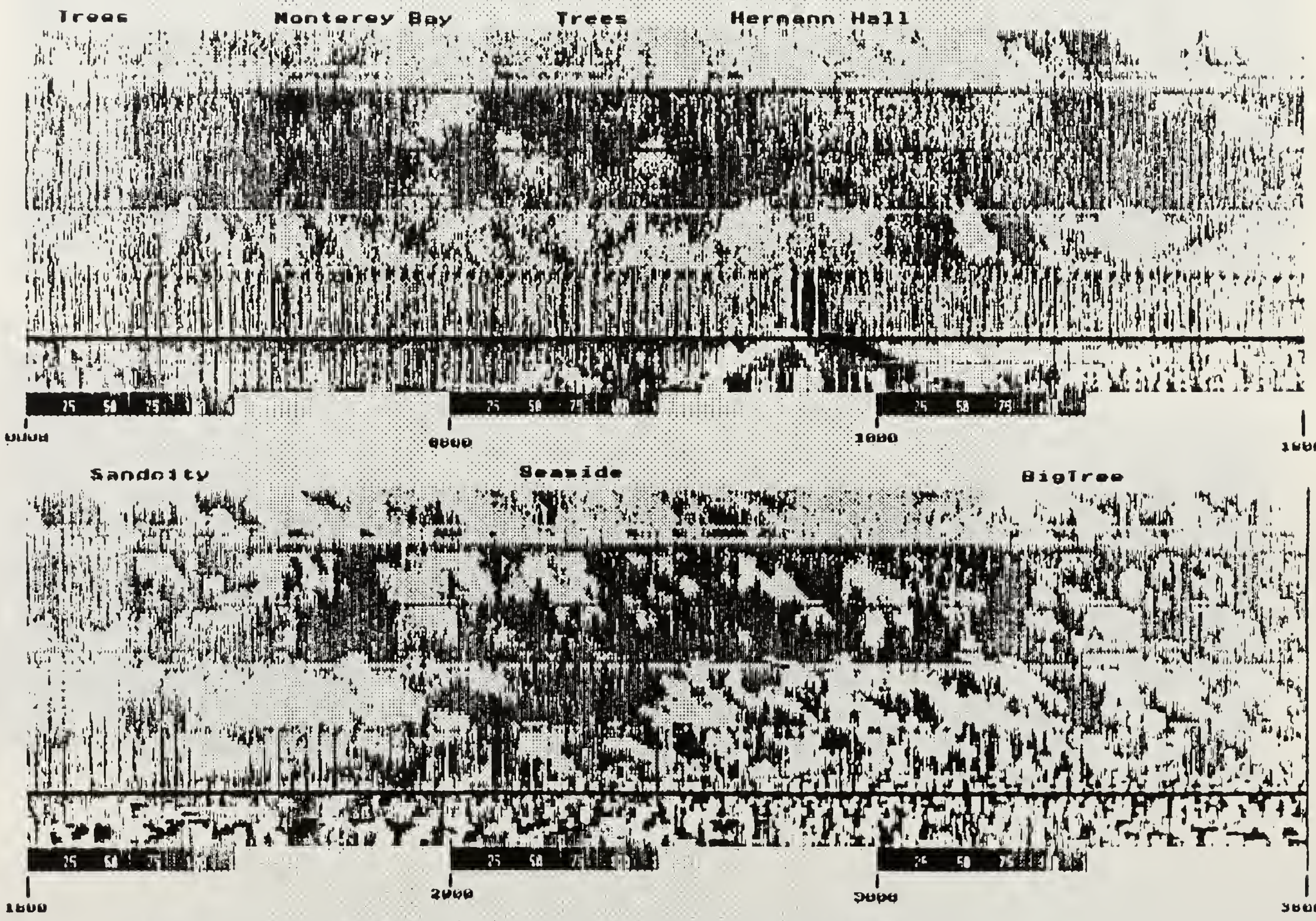


Figure 13. 72 degrees of the scanner rotation. Hexadecimal offset of the beginning of each 12 degree frame is indicated Direct histogram specification has been applied to enhance terrain features.

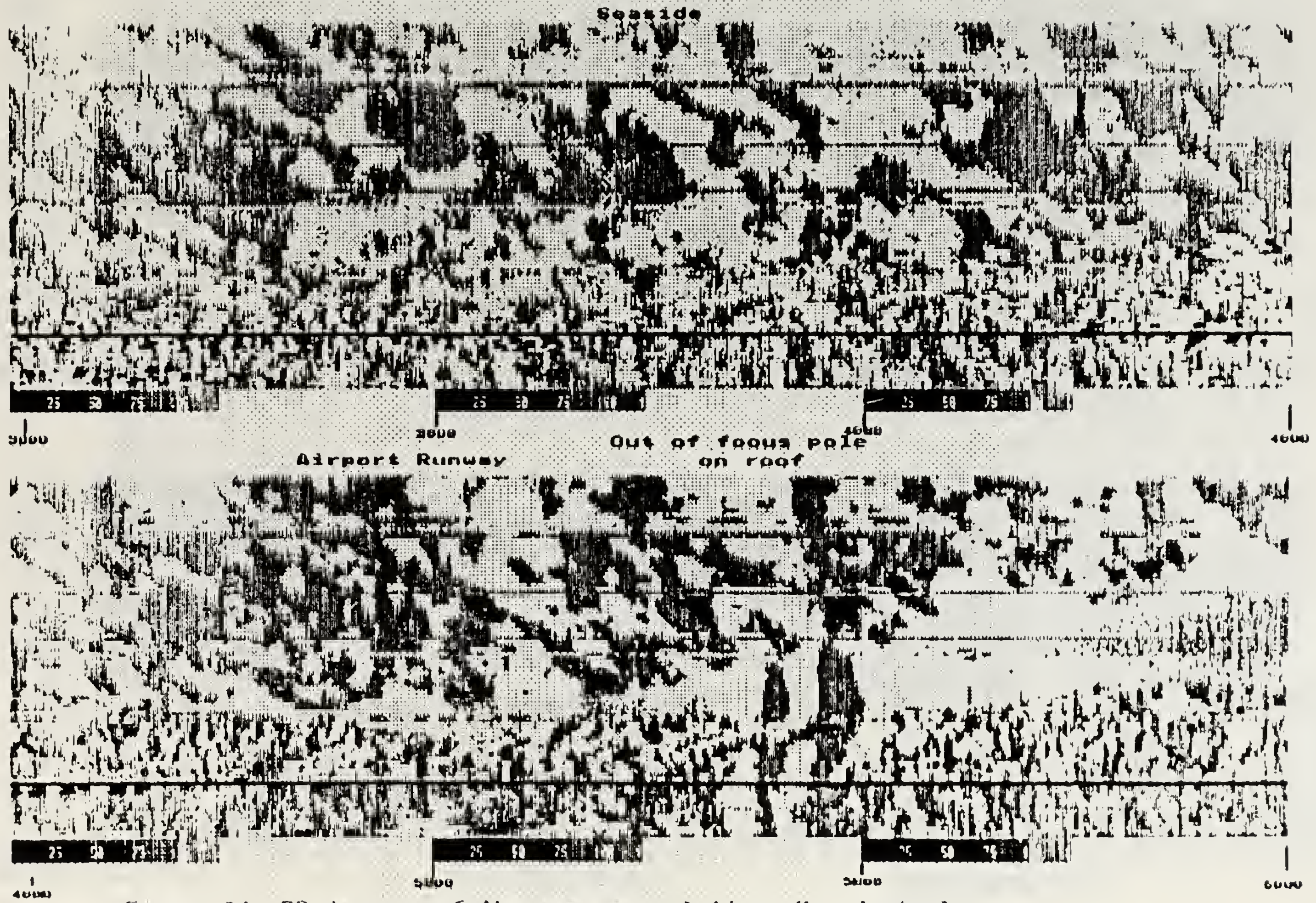


Figure 14. 72 degrees of the scanner rotation. Hexadecimal offset of the beginning of each 12 degree frame is indicated Direct histogram specification has been appliedto enhance terrain features.

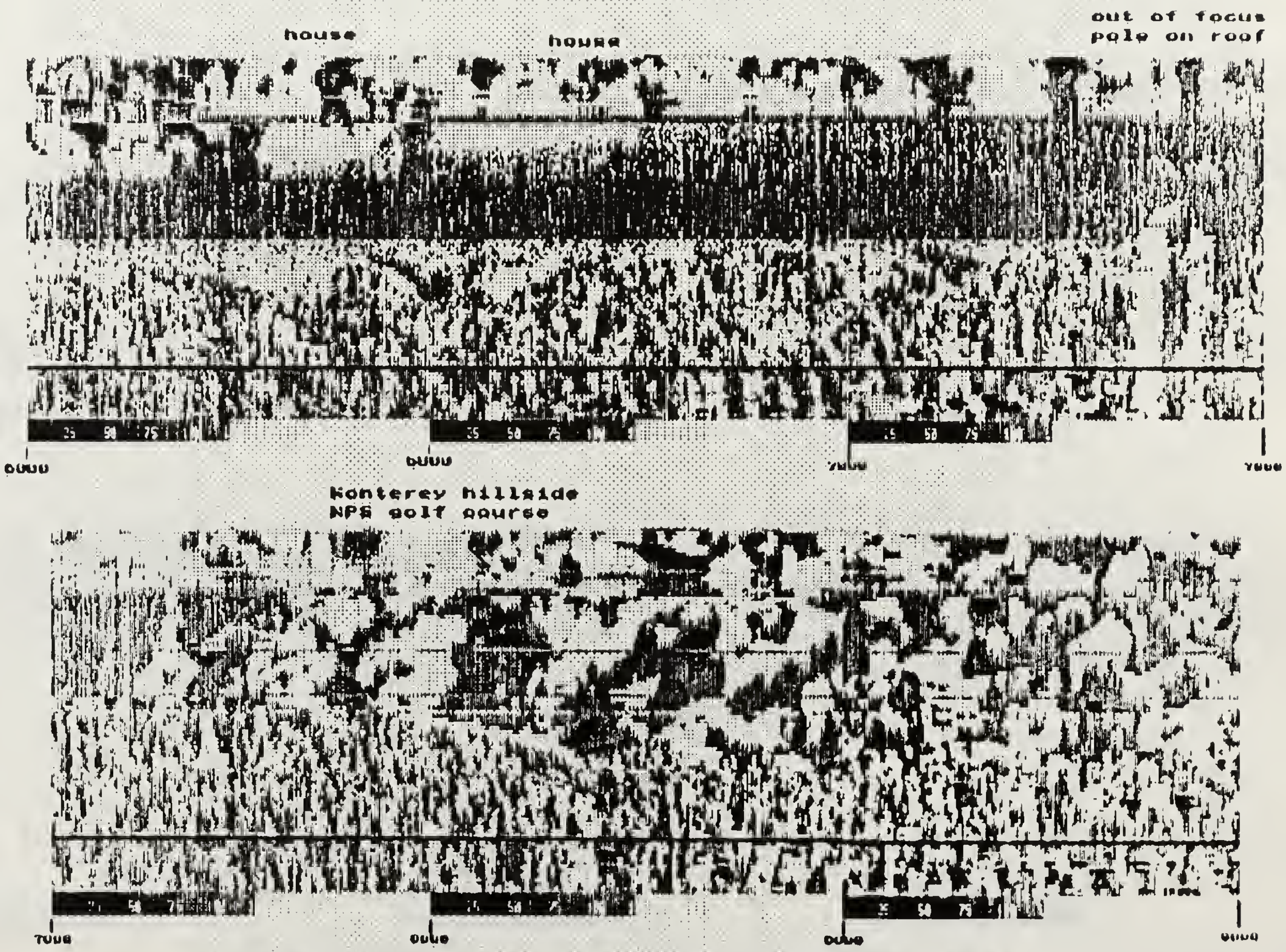


Figure 15. 72 degrees of the scanner rotation. Hexadecimal offset of the beginning of each 12 degree frame is indicated Direct histogram specification has been applied to enhance terrain features.

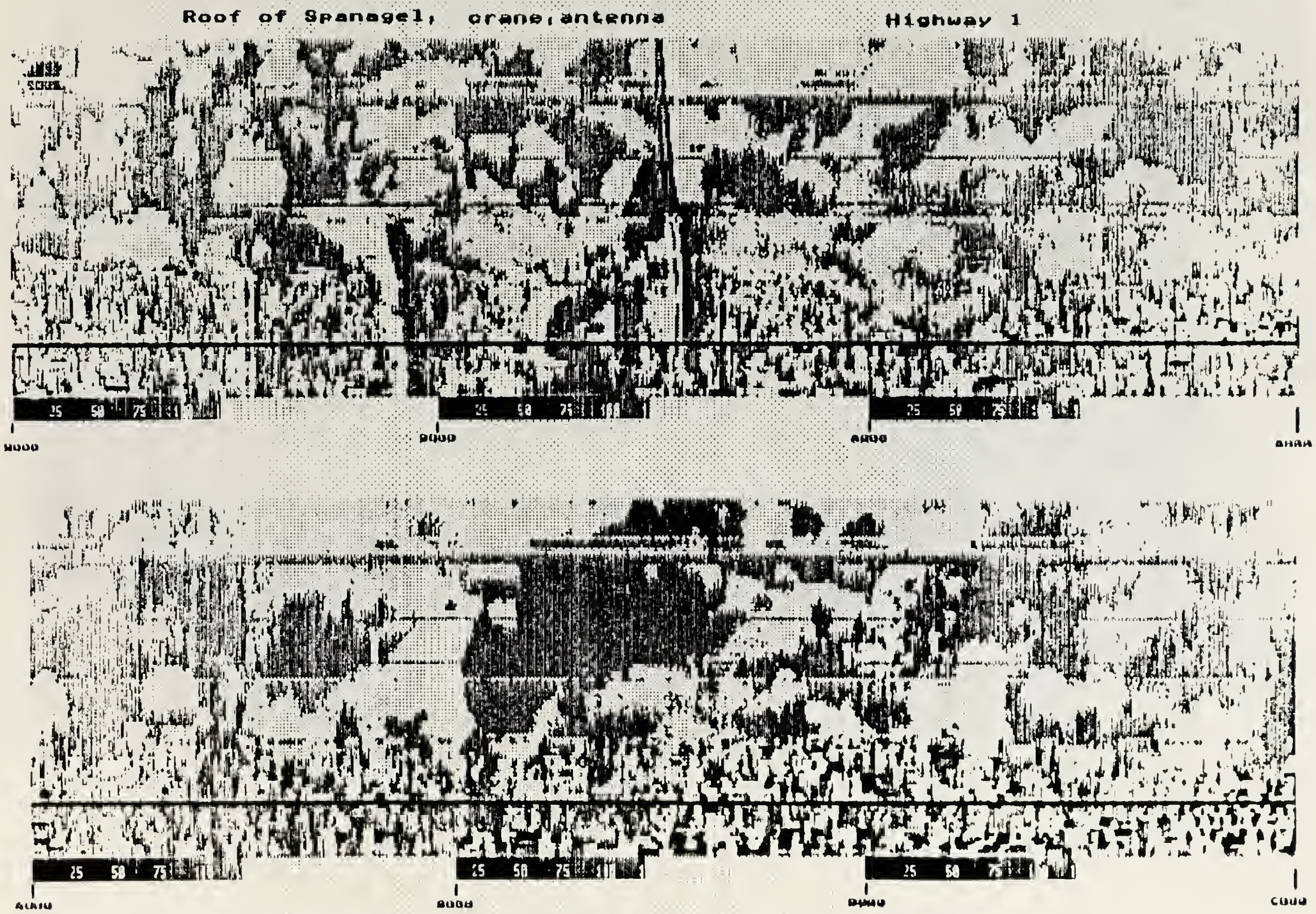


Figure 16. 72 degrees of the scanner rotation. Hexadecimal offset of the beginning of each 12 degree frame is indicated Direct histogram specification has been applied to enhance terrain features.

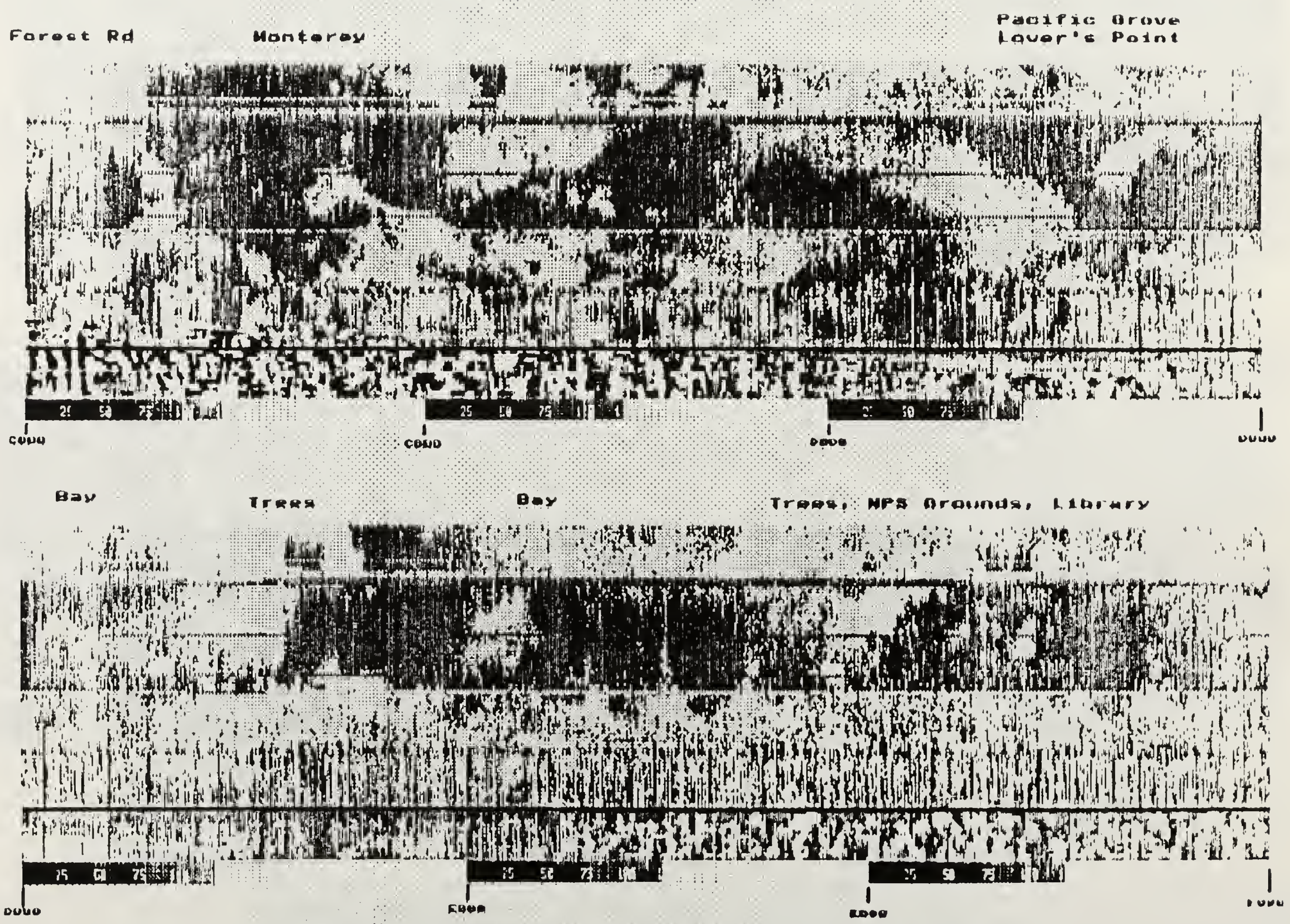


Figure 17, 72 degrees of the scanner rotation. Hexadecimal offset of the beginning of each 12 degree frame is indicated Direct histogram specification has been appliedto enhance terrain features.

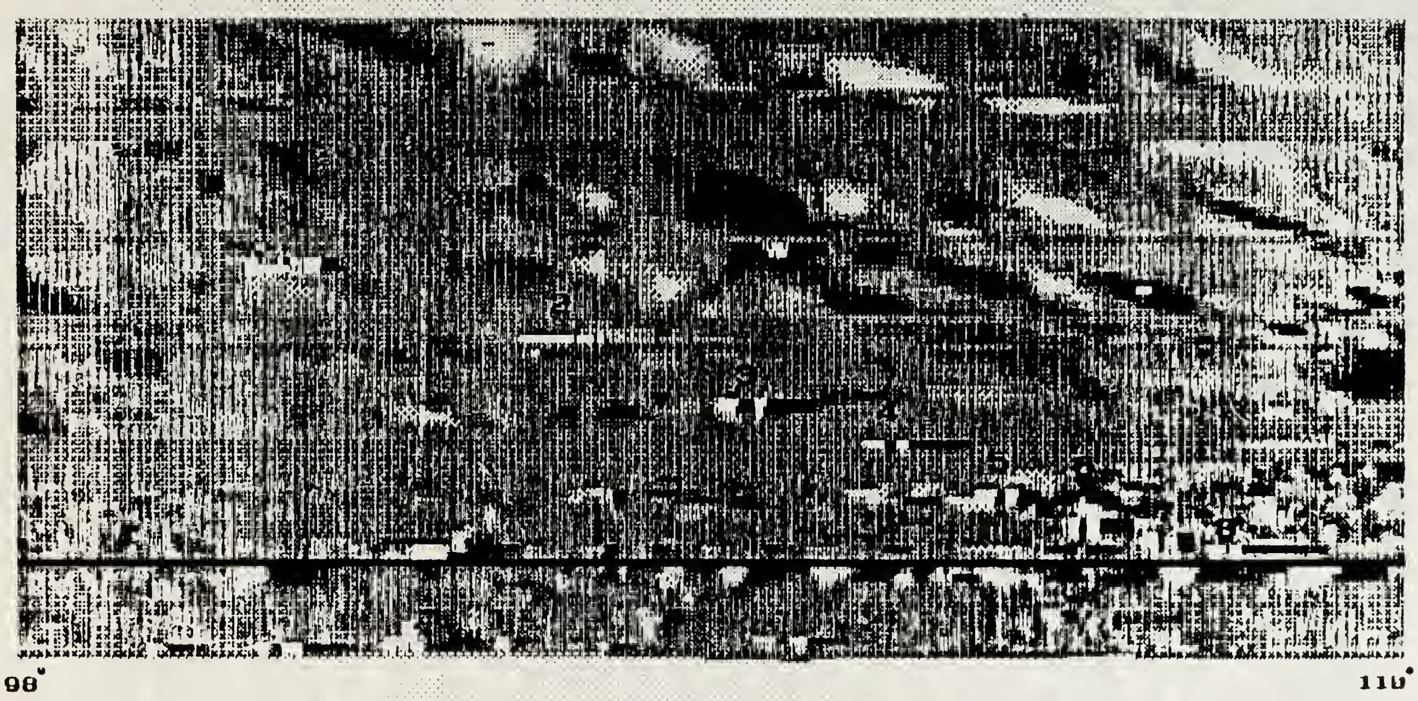


Figure 1B. US Air PropJet Passenger Aircraft Landing at the Monterey Airport.

IV. CONCLUSIONS AND RECOMMENDATIONS

A number of hardware and software developments have been brought to the NPS-IRSTD system which made possible the acquisition and display in real time of the output data in image form. Improvements to the frame grabber interface circuitry now allow a specific sector to be displayed and refreshed after each scanner rotation. Direct interfacing between the frame grabber interface board and the host computer AT bus enables each load operation of the frame grabber's memory buffers to be effected under software control. In order to greatly increase the speed of all signal processing tasks required prior to the video display of IRSTD images, the FORTRAN software was completely rewritten in assembly language and compiled with a 386 DOS Extender so that the extended memory buffers of the frame grabber board could be accessed directly. The assembly language software in itself runs at a much faster speed than its FORTRAN counterpart. In addition to improvements in speed, the assembly language program has the additional advantage of controlling the board operations directly, without requiring a device driver. Therefore, operations are not limited to those supported by the device driver and can be optimized for speed. The FORTRAN program on the other hand must deal with the frame grabber board via a device driver which is required by the DT-IRIS FORTRAN support library. As a result, the assembly language software can display a six degree pseudo-color sector in the scanner rotation with a refresh rate up to approximately 1.5 second during high speed data playback.

The ability to display IRSTD data in real time has opened the way to rapid developments in data acquisition and analysis. Interesting scene features can be located quickly anywhere in the rotation and targets can be followed from one rotation to the next with the use of the arrow keys on the computer keyboard. What would previously have taken hours to perform can now be done in a matter of seconds. A procedure for acquiring, enhancing and printing the video output using commercial image processing software packages has been described in detail.

A number of recommendations come to mind in order to improve the performance of the LOADUP program and the hardware interface.

A. HARDWARE IMPROVEMENTS

First of all, the second array should be made operational. This would allow comparisons between the images produced by the two arrays. The arrays are plagued by a relatively large number of dead or low responsivity detectors. The lag array has the unfortunate disadvantage of having a dead detector directly in line with the horizon. A number of methods have been tried to enhance the image output of the low detectors with limited success. One recourse in producing cleaner images would be to fill in the blank stripes in the image with the output from the other array.

A second hardware improvement would be to install the fiber optic link between the roof and the second floor. This will eliminate some of the crosstalk and noise in the signals sent down from the roof and will improve the reliability of the recordings.

A number of minor improvements are needed to improve the image display of the output data. As it stands now, the digitization rate is fixed to accommodate the recording weaknesses of the AMPEX HBR3000i tape recorder. The rotation rate is set as close as possible to the digitization rate, but the rotation rate of the mechanical device varies greatly even during one revolution. For the moment, the playback of data is linked to the digitization clock and the 1/180 pulses. This introduces jitter in the image display. To compensate for this, the playback of data should be linked with the position sensor so that the next position pulse gates the digitization signal. This would in effect prune the data so that excessive digitizations when the scanner is lagging in speed are discarded and that the number of digitizations analysed in a rotation is constant. This would allow images to be compared from rotation to rotation without jitter in the number of digitizations from the end of rotation pulse. Another possible source of jitter is the end of rotation (EOR) pulse itself. The EOR is derived from a slow photocell, so the position of the pulse edge varies. A modification to the circuitry is required to replace this pulse with the next sector pulse after it.

The source of the noise structure in the bottom part of the image must be investigated and the problem corrected. This is a prerequisite to further studies of infrared background scenes using the NPS-IRSTD.

The 5 Mhz clock can be divided by 2 to allow digitization of data at one half the rate of the original design. Since signals are oversampled by a factor of three, no information would be lost. As a consequence, the recording time would be doubled and more proportioned twelve degree images would be displayed in real time as opposed to six degree images. An alternative would be to divide the clock on playback so that all the data is recorded but only half of the data is sent for display.

The tape recorder allows digital control of its servo controls. Eventually, a hardware interface and software program should be developed to control the playback of a desired footage of tape from the computer.

B. SOFTWARE IMPROVEMENTS

The LOADUP program unscrambles the data from both arrays even though only one of the arrays is connected. Only data from the lag array is displayed, however. A minor change to the program needs to be made so that either one of the arrays can be displayed at the pressing of a key on the computer keyboard.

The program can be expanded to perform more signal processing tasks such as filling the dead or damaged detector bands. FORTRAN routines to correct the image from the effect of the low responsivity detectors have only had limited success. More developments in this area are needed before an assembly language routine can be designed.

A protected mode assembly language program to transfer the data from a frame buffer to the computer SuperVGA display in the 1024 by 768 pixel, 256 color mode would greatly accelerate the image enhancement and analysis process. Saving images to raw data 256 kilobyte files using IRIS-Tutor would no longer be needed. Images could be captured from the computer screen using a commercial capture program and saved to the more compressed PCX format thus saving enormous amounts of disk space. VGAIPS would no longer be needed as an interim step to display the images to the computer screen. The transfer routine could be built in the LOADUP program to permit real time acquisition of image files without interruptions to the display of images by the frame grabber.

Finally, the PROT 386 DOS Extender could be made compliant with DPMI and VCPI standards so that it could be used inside multitasking environments such as Deskview or Windows.

APPENDIX A

Protected Mode Operation on the 80386 Computer

A. BACKGROUND

The Intel 80386 and 486 computers are the most recent offsprings of a tremendous cumulative investment in the 86 family of processors. This enormous investment by computer and associated hardware manufacturers, and operating system and application software designers has essentially guaranteed the compatibility of the 86 family as it evolved from one generation to the next. One of the most compelling factors however comes from the fact that more than 60 million computers are currently running Microsoft software based on this family of processors. Perhaps the most striking example of this can be found in the overwhelming popularity of the MS-DOS operating system. The symbiosis which exists between Microsoft and Intel has not only impacted on the software industry but has placed high demands on the hardware design of the processors themselves. As a result, the 80386 features three modes of operation, Real mode, Virtual 86 mode and Protected mode, the first two of which are only there to ensure backward compatibility with the software written for earlier processors in the family.

Unfortunately, under the DOS paradigm the 80386 is restricted to Real mode operation and some of its most advanced features are not available. These features include flexible segmentation, privilege protection, multitasking and paging. Whenever the 80386 is Reset or powered up, it begins executing in Real mode and to remain compatible with the earlier 8086 processor, one of the address lines is disabled to prevent memory access above 1 megabyte. In order to alleviate this problem and to benefit from the full power of the 80386 processor while retaining compatibility with software dependant upon MS-DOS and the BIOS, an increasingly popular recourse has been to turn to DOS Extenders. As their name implies, DOS Extenders are extensions to the operating system which allow application software for the 80386 to operate in Protected mode. While doing so, DOS Extenders basically have to play the role of the operating system and provide a mechanism for handling interrupt-driven input/outputs as well as allowing calls to DOS and the BIOS until safe return to Real mode is reached.

In order to appreciate the advantage of running programs in Protected mode and the role of DOS Extenders, a certain amount of background knowledge with its associated terminology has to be presented. This section briefly covers some of the fundamental concepts surrounding Protected mode operation on a 386 computer. The material in this section comes mainly from the Intel 386 DX Programmer's reference manual [Ref. 14], the voluminous book by Crawford and Gelsinger: Programming the 80386 [Ref. 15] and the book by Turley: Advanced 80386 Programming Techniques [Ref. 16].

B. PROTECTION MECHANISMS

The 80386 has enough processing power to run several applications at once. This ability is refered to as multitasking. Each program in a multitasking environment constitutes a separate task. Although tasks appear to run simultaneously, they really only share the processor in sequence. Tasks all share the same computer resources: processor, memory space and peripherals. Therefore protection mechanisms must exist in order to ensure that each task is protected from every other task. For example, two programs running at the same time must occupy different address spaces; otherwise they would interfere with each other. If protection were implemented in software, by the operating system for instance, endless time-consuming checks would have to be made to ensure that commonly shared resources are not requested simultaneously. This would greatly impact on performance and application programs would appear to run sluggishly. To facilitate the operating system's task, protection mechanisms are enforced in hardware. Hardware protection and allocation of memory space is generally refered to as Memory-Management whereas I/O-Management supports the allocation and protection of peripheral devices.

1. Memory-Management

Memory-Management is a key protection mechanism. It consists of two parts: protection and address translation. Protection is used to prevent different tasks from accessing the memory space belonging to other tasks including the operating system. Address translation consists of segmentation and paging.

a. Memory addressing

The memory space on a 80386 based computer is divided into segments. Addresses are referenced by using a pointer which consists of two parts: a segment part and a 32-bit offset. The segment part is loaded into one of six available segment registers

CS,DS,ES,FS,GS and SS and indicates where a segment begins. The offset part provides a displacement to a specific byte within a segment. Since 32-bit offsets can be specified, the maximum size of a segment can therefore be 4 gigabytes. Usually, programs work with only a few segments, a Code segment with a base pointer in the CS register, a Data segment with a base pointer in the DS or ES register and a Stack pointed to by the SS register.

Addresses formed by combining the segment and offset parts are called virtual or logical addresses. They are called so because they do not correspond directly to a physical memory location but must be translated by a mapping function into an actual physical address. Address translation provides a form of protection. The mapping function can prevent certain physical locations from being accessed, deny memory access to some tasks or detect invalid or non-existant physical address references. Additionally, it can also map virtual addresses into a memory space much larger than the physical memory available by redirecting the address references to disk; a technique known as Virtual Memory Access.

b. Segmentation and Paging

Segments simplify the translation process by reducing the amount of information required to address physical memory. Through segmentation, the mapping function used for address translation can be defined for complete blocks of memory as opposed to requiring information for individual bytes. Segments are defined by three parameters: the base address, the segment limit which defines the size of the segment and the segment attributes which indicate the level of protection, whether or not the segment can be read from, written to or executed as a program. All segment information is contained in Segment Descriptor Tables. There are two types: the Global Descriptor Table (GDT) and the Local Descriptor Tables (LDTs). Each segment is defined by an entry in one of the descriptor tables. Each task has its associated LDT. The operating system sets up and maintains a GDT for the whole system. Segment information is stored in the descriptor tables by loading the appropriate registers, GDTR or LDTR. The descriptor tables are stored in a memory area protected by the operating system but which can be accessed by the memory-management hardware to control memory access. Paging differs from segmentation only in the way in which virtual memory is mapped. The process of memory translation is a two step process. First, the virtual address is converted into a linear address though segmentation. Then, the paging hardware completes the process by mapping the linear address to a physical address. Paging organizes memory in fixed blocks of 4 kilobytes called pages. When an address is issued by a software program, it is translated into a linear offset into a page reference. If the page exists in memory, the hardware accesses the physical address; otherwise it issues an exception to the operating system. The operating system then responds to the exception by loading the page from disk to memory. Paging can be disabled by the operating system in which case segmentation translates the logical address to a physical address directly.

c. Protection

Protection is a hardware mechanism by which the various tasks in a multitasking environment can be physically separated from each other even though they all share the same resources: processor, memory, display monitor, disks and other periferals. This is accomplished in conjunction with segmentation and paging. There are five protection checks:

(1) Type checking is used to verify if a segment is readable, writeable or executable;

(2) Limit checking prevents programs from accessing memory outside their assigned segments by attempting to form memory addresses with an offset part that is larger than the segment size;

(3) Restriction on the addressable domain is imposed by ensuring that the privilege level of the requesting task is equal or higher than the privilege level of the segment to be accessed;

(4) Restriction on the procedure entry points is ensured through privilege checking whenever control transfers between segments are requested by JMP, CALL, RET, INT and IRET instructions. Control transfers are executed by a procedure called task switching via one of four kinds of gates: call gate, trap gate, interrupt gate, and task gate.

(5) Restriction on part of the instruction set is used to prevent privileged instructions and certain sensitive input/output activities reserved for the operating system from being used by tasks with insufficient privilege level.

Protection check information is stored in reserved fields in each segment descriptor (GDT or LDT entries). The last three checks are performed via a mechanism which recognizes four privilege levels numbered 0, 1 , 2 and 3.

2. I/O Management

Peripheral devices and communication serial interfaces are controlled through Input/Output (I/O) ports. Because all tasks are liable to require access to these ports, special mechanisms are in place in the 386 microprocessor to manage I/O operations. There are two ways to address a port and the appropriate method for a given periferal device is decided by its designer. The 386 microprocessor reserves a separate I/O address space where I/O port addresses can be located. Port addresses in this reserved memory space can be accessed by using the processor's IN and OUT instructions. These instructions enable the M/IO# pin on the processor chip which is used in the decoding of the port address. In this way, port addresses are kept distinct from the address space for physical memory. A special field in the segment descriptor for the I/O address space is used to prevent I/O pages from being relocated by the segmentation and paging mechanisms. Protection is ensured by verifying a field in the processors EFLAGS register and the I/O permission bit map in the Task State Segment (TSS) descriptor table. A second method for accessing peripheral device ports is refered to as memory-mapped I/O. Devices whose ports can respond to bus accesses in the same manner as memory components can be mapped in the physical address space of the processor and be accessed by any instructions which reference memory. In this case, the port addresses are subject to all the protection mechanisms described earlier.

3. Exceptions and Interrupts

Whenever a peripheral device, (printer, disk controller, keyboard or other), requires the attention of the processor, it triggers an interrupt signal which is immediately attended to by the processor. The current procedure is suspended at the end of the instruction being executed and control is transfered to a task or procedure called a handler which handles the interrupt. Exceptions are similar to interrupts in that they are handled in the same way. The difference is that interrupts occur randomly at the request of a peripheral whereas exceptions are the direct outcome of an instruction being executed. There are 16 different exceptions ranging from divide by zero error and overflow errors to page faults and coprocessor errors. Exception 13 is for general protection errors and acts as a catch-all for all errors which do not have their own handler. Exceptions are further classified as faults, traps and aborts depending on how they are handled. Software interrupts to DOS and the BIOS which make use of the INT instruction are exceptions rather than interrupts.

In Real Mode, the address of the service routine is referred to as a vector and is located in a table called a vector table. Exceptions and interrupts are handled by the processor on a priority basis. In Protected Mode, the priority information, the interrupt vector and the task which handles the interrupt/exception are specified in the Interrupt Descriptor Table (IDT). Entries in the IDT are made by writing to the IDTR register.

4. 386 Protected Mode Initialization

Upon RESET, the 80386 begins executing in Real Mode. In Real Mode, all 32 bit registers are available except the Task Register (TR) and the Local Descriptor Table Register (LDTR) which are used for multitasking in Protected Mode. All of the 80386 enhanced instruction set can be used including all privileged instructions and I/O instructions since·the only task which can be executed at a given time is given the highest privilege level. Memory addressing in Real Mode is completely different from the method described earlier for Protected Mode. In Real Mode, memory segment descriptors are not used. Linear addresses are limited to 1 megabytes and logical offsets cannot be greater than 64 kilobytes regardless of whether or not 16 bit or 32 bit registers are used for addressing. The A20 address line is disabled for compatibility with the 8086 computer. Page translation, multitasking and page level protection are not implemented. Furthermore, interrupt and exception handling is performed differently. In Real Mode the interrupt vector consists in a segment:offset pair which points directly to the address of the first instruction of the service routine. The interrupt vector table is at a fixed address and cannot be relocated. In Protected Mode the IDT can be loaded anywhere by the paging mechanism and its size and content can be changed.

The procedure for switching from the RESET state or Real Mode to Protected mode is covered in a few paragraphs in the Intel literature [Ref. 14]. Technically, Protected Mode is entered when the PE bit of the CR0 register is set. The actual process is quite complicated, however, and a very careful and much more detailed procedure must be followed to ensure a successful transfer between the two modes [Ref. 16]. As a minimum, the GDT and IDT must be created before entering Protected Mode. A number of difficulties can arise if precautions are not taken. First, problems can occur between the time the IDT is loaded and Protected Mode is enabled since during this short period the interrupt table is inconsistent with the current operating mode. Second, arrangements must be made to ensure that the code which performs the switch to Protected Mode occupies the same linear space after enabling paging as it did before. Third, the instruction queue must be flushed since instructions remaining in the queue after the switch to Protected Mode were formed with the addressing mechanism consistent with Real Mode operation and therefore are no longer valid. The complete initialization procedure can be implemented in a few pages of assembly language code [Ref. 17]; however, once in Protected Mode, most assembly language applications require support similar to that provided by DOS's interrupt services to access and control the hardware and because of this, it is not advantageous to switch over to Protected Mode unless a mechanism to handle hardware interrupts and exceptions is implemented at the same time.

The above tasks are performed typically by Protected Mode operating systems such as UNIX or OS/2. Because DOS Extenders are not fully fledged operating systems they must deal with the additional complication of remaining compatible with the old-style DOS and BIOS interrupt and exception handling mechanisms. Memory addressing and interrupt handling in Protected Mode and in Real Mode differ drastically, to the point of conflicting with each other. Because of this, the prospect of taking advantage of Protected Mode features while retaining the services and exception handling routines provided by the BIOS and DOS seems bleak. There are two avenues for success however. One is to handle interrupts and BIOS calls by returning temporarily to Real Mode at every occasion. The other is to enable multitasking and run BIOS calls as a Virtual-86 Task. Virtual-86 Mode is the mode of operation of the earlier 8086 microprocessor. Virtual-86 tasks can run in a multitasking environment and can take advantage of the hardware support provided for multitasking on the 80386. This is the method employed by the PROT 386 DOS Extender used in this thesis. In fact the PROT 386 DOS Extender runs DOS and the BIOS as a Virtual-86 Task [Ref. 19].

APPENDIX B

The FORTRAN: DISPLAY.for program

```
C DISPLAY.FOR
C MS-FORTRAN 4.1
C  by Maj. J.D. Bernier
C  January 1991
C  Last revision: 13 September 1991

$LARGE
      COMMON IMG,IBUFF

  2    FORMAT(/)
       WRITE(*,2)
C
      CALL ISINIT
C
      WRITE(*,*) 'CLEAR ALL BUFFERS'
      DO 200 IBUF=1,15
        CALL ISFCLR(IBUF)
 200   CONTINUE

      CALL COLORS
      CALL SCALE
C
1      CONTINUE
       DO 100 IBUFF = 5,15
C
C  LOAD A FULL BUFFER
      CALL LOAD

C  UNSCRAMBLE 1ST 3 DEGREE IMAGE
      IMG=2
      CALL UNSCRB(IMG)
      CALL OFFSET
      CALL RESP

C  UNSCRAMBLE 2ND 3 DEGREE IMAGE
      IMG=3
      CALL UNSCRB(IMG)
      CALL OFFSET
C      CALL RESP

C  CREATE 6 DEGREE IMAGE
      CALL SIXDEG

C  DISPLAY FINAL RESULT
      CALL ISFCOP(4,IBUFF)
      CALL ISOTFR(IBUFF)
      CALL ISDISP(1)
      WRITE(*,*) 'DISPLAYING RESULT IN BUFFER ',IBUFF
C

100 CONTINUE
      GOTO 1
      CALL ISEND
      END

C ************************ SUBROUTINES *******************************
C
C *** LOAD ***
C
      SUBROUTINE LOAD
      EXTERNAL BUSY
      WRITE(*,*) 'LOADING ONE FRAME INTO BUFFER 0'
      WRITE(*,*) 'WAITING FOR EOR SIGNAL'

      CALL ISINFR(0)
      CALL ISSETR(0,0,512,512)
      CALL ISRDEP
      CALL BUSY
      WRITE(*,*) 'BUFFER LOADED'
      RETURN
      END


C *** UNSCRB  ***

      SUBROUTINE UNSCRB(IMG)
      INTEGER*2 SCRAM(23040)
      INTEGER*2 UNSCRAMB(23040),HALF(11520),ARRAY(90,128)
      INTEGER IROW,IAROW,IBROW,ICROW,IDROW,IEROW
      INTEGER IX,IA,JA,KA,ISECT,IBUFF,IBUFM1,IMG,IS,IE
      INTEGER*2 A(512),B(512),C(512),D(512),E(512)
      EQUIVALENCE(A,B,C,D,E)
      COMMON IBUFF
```

```
C  THE ORDER ARRAY CONTAINS THE UNSCRAMBLING SEQUENCE
   DIMENSION ORDER(180)

   DATA (ORDER(I),I=1,10) /2,17,32,47,62,77,92,107,122,137/
   DATA (ORDER(I),I=2,20) /152,167,3,18,33,48,63,78,93,108/
   DATA (ORDER(I),I=3,30) /123,138,153,168,4,19,34,49,64,79/
   DATA (ORDER(I),I=4,40) /94,109,124,139,154,169,5,20,35,50/
   DATA (ORDER(I),I=5,50) /65,80,95,110,125,140,155,170,6,21/
   DATA (ORDER(I),I=6,60) /36,51,66,81,96,111,126,141,156,171/
   DATA (ORDER(I),I=7,70) /7,22,37,52,67,82,97,112,127,142/
   DATA (ORDER(I),I=8,80) /157,172,8,23,38,53,68,83,98,113/
   DATA (ORDER(I),I=9,90) /128,143,158,173,9,24,39,54,69,84/
   DATA (ORDER(I),I=10,100) /99,114,129,144,159,174,10,25,40,55/
   DATA (ORDER(I),I=11,110) /70,85,100,115,130,145,160,175,11,26/
   DATA (ORDER(I),I=12,120) /41,56,71,86,101,116,131,146,161,176/
   DATA (ORDER(I),I=13,130) /12,27,42,57,72,87,102,117,132,147/
   DATA (ORDER(I),I=14,140) /162,177,13,28,43,58,73,88,103,118/
   DATA (ORDER(I),I=15,150) /133,148,163,178,14,29,44,59,74,89/
   DATA (ORDER(I),I=16,160) /104,119,134,149,179,164,15,30,45,60/
   DATA (ORDER(I),I=17,170) /75,90,105,120,135,150,165,180,1,16/
   DATA (ORDER(I),I=18,180) /31,46,61,76,91,106,121,136,151,166/

   CALL ISRSET
   CALL ISSETR(0,0,512,512)
   CALL ISOUTS(2)
   CALL ISOTFR(4)
   CALL ISDISP(1)

   ISECT = 0
   IF (IMG.EQ.2) IS=0
   IF (IMG.EQ.2) IE=135
   IF (IMG.EQ.3) IS=180
   IF (IMG.EQ.3) IE=315
   IF (IMG.EQ.2) WRITE(*,*) 'UNSCRAMBLING 1ST 3 DEGREE IMAGE'
   IF (IMG.EQ.3) WRITE(*,*) 'UNSCRAMBLING 2ND 3 DEGREE IMAGE'
   DO 430 IROW = IS,IE,45
      K = 0
      CALL ISGETP(0,IROW,0,23040,SCRAM)
      DO 410 J=0,23040,180
        DO 420 I=1,180
              K=ORDER(I)
              M=I+J
              N=K+J
              UNSCRAMB(N) = SCRAM(M)
420       CONTINUE
410     CONTINUE
```

```
        KA = 90
        DO 500 IA = 1,128
           DO 510 JA = 1,90
               KA = KA + 1
               ARRAY(JA,IA) = UNSCRAMB(KA)
 510       CONTINUE
        KA = KA + 90
 500  CONTINUE
C
          KA = 0
          DO 610 JA = 1,90
               DO 600 IA = 1,128
                 KA = KA + 1
                 HALF(KA) = ARRAY(JA,IA)
 600           CONTINUE
 610      CONTINUE
          CALL ISSETR(0,ISECT,90,128)
          CALL ISPUTR(1,HALF)
          ISECT = ISECT + 128

 430  CONTINUE
C  THIS SECTION EXPANDS THE DATA FROM 90 LINES TO 450 LINES.
C
     IROW = 0
     DO 800 IROW = 0,89
           CALL ISGETP(1,IROW,0,512,A)
           IAROW = 5*IROW
           CALL ISPUTP(IMG,IAROW,0,512,A)
           IBROW = 5*IROW + 1
           CALL ISPUTP(IMG,IBROW,0,512,B)
           ICROW = 5*IROW + 2
           CALL ISPUTP(IMG,ICROW,0,512,C)
           IDROW = 5*IROW + 3
           CALL ISPUTP(IMG,IDROW,0,512,D)
           IEROW = 5*IROW + 4
           CALL ISPUTP(IMG,IEROW,0,512,E)
 800  CONTINUE
     WRITE(*,*) 'BUFFER UNSCRAMBLED'
     RETURN
     END
C
C *** OFFSET ***

        SUBROUTINE OFFSET
          COMMON IMG
          CALL ISSETR(150,0,75,512)
          CALL ISOFFC(IMG,-3,IMG)
          CALL ISSETR(0,0,512,512)
          RETURN
          END
```

*** SCALE ***

```
     SUBROUTINE SCALE
     INTEGER*2 IBACK(15360), NUM(2), NUM1(3)
C
   2 FORMAT (/)
     WRITE(*,*) 'CREATE COLOR SCALE AT BOTTOM OF SCREEN'
C   THIS SECTION ADDS LABELS AND A COLOR SCALE ACROSS THE
BOTTOM
C   OF BUFFER 2.
C
        IFR=2
      DO 110 I=0,29
        DO 120 J=0,510,2
          IBACK(I*512+J+1)=J/2
          IBACK(I*512+J+2)=J/2
 120      CONTINUE
 110    CONTINUE
      CALL ISPUTP (IFR,450,0,15360,IBACK)
      CALL ISSFNT (0)
      NUM(1)=50
      NUM(2)=53
      CALL ISGPOS (460,45)
      CALL ISTEXT (IFR,2,NUM)
      NUM(1)=53
      NUM(2)=48
      CALL ISGPOS (460,95)
      CALL ISTEXT (IFR,2,NUM)
      NUM(1)=55
      NUM(2)=53
      CALL ISGPOS (460,145)
        CALL ISTEXT (IFR,2,NUM)
      NUM1(1)=49
      NUM1(2)=48
      NUM1(3)=48
      CALL ISGPOS (460,192)
      CALL ISTEXT (IFR,3,NUM1)
      NUM1(1)=49
      NUM1(2)=50
      NUM1(3)=53
      CALL ISGPOS (460,242)
```

```
        CALL ISTEXT (IFR,3,NUM1)
        NUM1(1)=49
        NUM1(2)=53
        NUM1(3)=48
        CALL ISGPOS (460,292)
        CALL ISTEXT (IFR,3,NUM1)
        NUM1(1)=49
        NUM1(2)=55
        NUM1(3)=53
        CALL ISGPOS (460,342)
        CALL ISTEXT (IFR,3,NUM1)
        NUM1(1)=50
        NUM1(2)=48
        NUM1(3)=48
        CALL ISGPOS (460,392)
        CALL ISTEXT (IFR,3,NUM1)
        NUM1(1)=50
        NUM1(2)=50
        NUM1(3)=53
        CALL ISGPOS (460,442)
        CALL ISTEXT (IFR,3,NUM1)
      RETURN
      END


C *** COLORS ***

      SUBROUTINE COLORS
      INTEGER*2 ICOLBA,ICOLBB,ICOLGA,ICOLGB,ICOLRA,IRD,IGR,IBL
      INTEGER*2 I,J,K,L,M,N,IBK,KRD
      WRITE(*,*) 'GENERATE DEFAULT OUTPUT COLOR TABLE'
C
    7 FORMAT (4(2X,I6))
C
      OPEN (35,FILE='COLORS.DAT')
C
      K=2
      IBL=0
      IGR=0
      IRD=0
      DO 20 I=0,79
        J=255-I
        CALL ISLDOV(K,J,IRD,IGR,IRD)
   20 CONTINUE
      DO 10 I=80,170
      ICOLBA=INT((I-80)*7.5)
      ICOLBB=INT((I-140)*10.8)*(-1)
      ICOLGA=INT((I-105)*6.7)
      ICOLGB=INT((I-160)*18)*(-1)
      ICOLRA=INT((I-130)*4.3)
      IF (I.GT.79 .AND. I.LE.115) IBL=ICOLBA
      IF (I.GT.115.AND. I.LE.140) IBL=ICOLBB
      IF (I.GT.140.AND. I.LE.170) IBL=0
      IF (I.GT.79 .AND. I.LE.105) IGR=0
      IF (I.GT.105.AND. I.LE.145) IGR=ICOLGA
      IF (I.GT.145.AND. I.LE.160) IGR=ICOLGB
      IF (I.GT.160.AND. I.LE.170) IGR=0
      IF (I.GT.79 .AND. I.LE.130) IRD=0
      IF (I.GT.130.AND. I.LE.170) IRD=ICOLRA
      IF (IBL.GT.255) IBL=255
      IF (IGR.GT.255) IGR=255
        IF (IRD.GT.255) IRD=255
        J=255-I
        WRITE (35,7) J,IRD,IGR,IBL
        k=2
      CALL ISLDOV(k,J,IRD,IGR,IBL)
  10 CONTINUE
    IRD=255
    IBL=0
    IGR=0
    DO 30 I=171,255
      J=255-I
      CALL ISLDOV(K,J,IRD,IGR,IBL)
  30 CONTINUE
    CALL ISOUTS (2)
    RETURN
    END

C *** RESP ***

    SUBROUTINE RESP
    DIMENSION    XRES(90)
    INTEGER*2    IROW1(1:512),IROW2(1:512)
    INTEGER    NFRAME
C
C   THIS SECTION CORRECTS FOR THE DIFERENT RESPONSES OF THE
C   DETECTORS
C   IN THE ARRAY
C
C THIS DATA SET  CONTAINS THE VALUES OF RESPONSIVITY MEASURED
C BY AYERS IN  HIS EVALUATION OF THE AN/SAR-8 FOR THE LAG ARRAY.
```

```
C Caution:
C in Ayers Thesis, detectors are numbered starting at 1 from the bottom!!!
C
    DATA XRES/2.219,2.840,2.952,3.161,3.330,3.213,3.082,2.882,2.970,
   *3.142,2.480,2.580,2.613,3.164,3.042,3.160,3.219,3.299,3.340,0.015,
   *3.110,3.267,3.149,3.101,3.031,3.141,3.055,3.102,2.911,3.058,3.231,
   *3.131,3.280,3.109,3.144,3.242,3.201,3.113,3.268,3.291,1.183,3.383,
   *3.261,3.233,3.303,3.404,3.452,3.450,3.410,3.281,3.361,3.150,3.030,
   *3.122,3.120,3.322,3.000,3.050,2.475,3.050,0.030,0.600,0.600,0.022,
   *0.020,3.140,3.270,3.202,3.028,2.802,3.119,3.391,3.305,3.262,3.166,
   *3.180,3.189,3.251,3.320,3.132,3.163,3.302,3.321,3.261,3.400,3.512,
   *3.189,3.382,3.433,0.020/

C  COMPUTE THE AVERAGE OF THE   RESPONSIVITY VALUES
C
    NFRAME=2
    AVGRES=0
    DO 410 I=1,90
      IF (XRES(I) .LT. 0.7*AVGRES/I) XRES(I)=AVGRES/(I-1)
      AVGRES=AVGRES+XRES(I)
 410 CONTINUE
    AVGRES=AVGRES/90
C
C  CORRECT EACH ROW OF VALUES BASED ON THE RELATIVE
RESPONSIVITY
C
    DO 420 I=0,445,5
        CALL ISGETP(NFRAME,I,0,512,IROW1)
        DO 430 J=1,512
        IROW2(J)=INT((IROW1(J)-128)*AVGRES/XRES((I/5)+1)+128)
        IF (IROW2(J) .GT. 255) IROW2(J)=255
 430   CONTINUE
      DO 440 K=0,4
          CALL ISPUTP(NFRAME,I+K,0,512,IROW2)
 440   CONTINUE
 420 CONTINUE
    RETURN
    END

C
C *** SIXDEG ***

C  THIS SECTION BUILDS A 6 DEGREE HORIZONTAL IMAGE IN FRAME
C  BUFFER 1 FROM THE 3 DEGREE IMAGES IN BUFFERS 2 AND 3.
C
C
```

```
   SUBROUTINE SIXDEG
   COMMON IBUFF
   INTEGER*2 IROW1(0:511),IROW2(0:511),ISIXA(0:255),ISIXB(0:255)
300 CALL ISDIVC(2,2,0)
   CALL ISDIVC(3,2,1)
   DO 310 J=0,449,5
        CALL ISGETP(0,J,0,512,IROW1)
        CALL ISGETP(1,J,0,512,IROW2)
     DO 320 K=0,511,2
       ISIXA(K/2)=IROW1(K)+IROW1(K+1)
       ISIXB(K/2)=IROW2(K)+IROW2(K+1)
320    CONTINUE
     DO 321 K=0,4
       L=J+K
         CALL ISPUTP(4,L,0,256,ISIXA)
         CALL ISPUTP(4,L,256,256,ISIXB)
321    CONTINUE
310 CONTINUE
   RETURN
   END
```

APPENDIX C

DT-2861 REGISTERS

INCSR1	Input Control Status Register 1	0250h	Controls the video input
INCSR2	Input Control Status Register 2	0252h	Controls the video input
OUTCSR	Output Control Status Register	0254h	Controls the video output
CURSOR	Cursor Register	0256h	Contains the cursor and line pixel position
INDEX	Index Register	0258h	Contains the LUT Index
XPAN	XPan Register	0258h	Controls X direction pans
INLUT	Input Look-Up table entry register	025Ah	Contains the input LUT entry

RLUT	Result Look-Up table entry register	025Ah	Contains the result LUT entry
YPAN	Ypan register	025Ah	Controls the Y direction pan
REDGRN	Red/Green Output Look-Up table register	025Ch	Contains the Red & green LUT entry
START	Start register	025Ch	Defines the sarting line and pixel
BLUE	Blue Look-Up table entry register	025Eh	Contains the Blue LUT entry
END	End register	025Eh	Defines the ending line and pixel

APPENDIX D

The LOADUP Protected Mode Assembly Language Program

```
comment +
Subroutine load6.pm
by Maj. J.D.Bernier
MASM 5.1
5 April 1991
Last Revision: 13 September 1991

This routine loads up, unscrambles and expands horizontally by a factor
of five, an image buffer on the DT-2861 Frame Grabber board.

Important:
loadup.pm switches to 386 Protected Mode and
will not run if other Extended memory managers such as QEMM.sys
or 386max.sys or if other DOS Extender programs are memory resident.

This routine is compiled and linked with the PROT Dos Extender
by Al Williams described in a two part article which appeared in
Dr Dobb's Journal, October and November 1990.
A few lines of example code are added for future reference
on the use of the PROT Dos Extender.
+

        PROT_DATA

message db "Running in 386 Protected Mode",13,10,"$"
rotmsg  db "Enter offset angle : ",13,10,"$"
endmsg  db "Program terminated normally by user",13,10,"$"
palette db "Do you wish to program the output LUT (y or n) ?",13,10,"$"
keybuf  db 5            ;establish a keyboard input buffer with
msg     db ?            ;length equal to 4 + CR
        db 5 dup(?)
        db 13           ;end buffer with a CR
hex     db "0123456789ABCDEF"
sectmsg db "Rotation Sector Offset: "
sector  db ?,?,?,?,"h",13,10,"$"
rotate  equ 0800h       ;moves displayed field by 6 degrees

;DT-2861 registers and buffer addresses
```

```
        INCLUDE DT-2861.INC

;JDR PR-10 port address

JDR     equu    300h

;sort routine unscramble order

        INCLUDE ORDER.INC

;LUT entries

        INCLUDE     LUT.INC

;operating modes (See DT-IRIS Reference Manual)

EPORTI      equ    0070h  ;MODE=111, input buffer 0
LDINLUT     equ    0020h  ;MODE=010
LDRLUT      equ    0030h  ;MODE=011
OSEL        equ    0000h  ;OSEL=000, BUSBUF=000, DISPLAY OFF
ALU         equ    0000h  ;ALU=1111,CARRYIN=0,BUSY=0,PASS=0,ALUM=0
SLINE       equ    0000h  ;START LINE=0, START PIXEL=0
ELINE       equ    0B67Fh;END LINE=360,END PIXEL=512
LOADBUF     equ    0080h  ;ALU=1111,BUSY=1,PASS=0,ALUM=0,CARRYIN=0
DISBUF      equ    2000h  ;set display buffer 2
BUS01       equ    0020h  ;set active bus buffers 0 and 1
BUS23       equ    0220h  ;set active bus buffers 2 and 3
s_row  equ     150        ;A/D converter offet, start row
height  equ    75         ;A/D offset, height

        PROT_DATA_END

        PROT_CODE
USER    PROC NEAR
        PROT_STARTUP
        pushad                  ;save all 32-bit registers on stack

;examples of some PROT functions

        CALL32F sel_code32, CLS     ;clear PC screen
        CALL32F sel_code32, CRLF    ;CR/LF
;       mov al, 'P'                 ;write characters to PC screen
;       CALL32F sel_code32, OUCH    ;from protected mode
;       mov al, 'M'
;       CALL32F sel_code32, OUCH
;       mov al, ' '
;        CALL32F sel_code32, OUCH
;        CALL32F sel_code32, CRLF
         mov ax, 21h
         mov PINTFRAME.VMINT,eax     ;BIOS call from protected mode
         mov edx, OFFSET message     ;This example uses DOS's call 09
         mov ah, 9                   ;to output a string to the screen
         mov ebx, OFFSET PINTFRAME
         VM86CALL

;fill frame buffer 0 with data from the order array
;(Debugging aid!!! keep for future reference)

;        mov ecx, 512             ;fill up buffer 0 with 180 rows of
                                  ;512 bytes taken form table "order"
;        mov eax, buff0
;        xor edi, edi
;fill:   mov ebx, OFFSET order
;        mov dl,BYTE PTR [ebx+edi]
;        mov BYTE PTR fs:[eax], dl
;        inc eax
;        inc edi
;        cmp edi, 180
;        jl short skip2
;        xor edi, edi
;        dec ecx
;skip2:  jnz short fill

;display first row of scrambled buffer
;        mov ebp, buff0
;        mov ecx, 512
;show1: mov al, BYTE PTR fs:[ebp]
;        call32f sel_code32, HEXOUT
;        inc ebp
;        loopne short show1
;        breakpoint

;initialize frame grabber board operation

         mov dx, OUTCSR
         in  ax, dx
         bts ax, 5                ;DISPLAY on
         out dx, ax

         mov ax, DISBUF           ;set display buffer 2
         mov dx, YPAN
         out dx, ax
```

```
        mov dx, INCSR1          ;stop current operation
        in  ax, dx
        btr ax, 5               ;clear PASS bit to terminate current
        out dx, ax              ;operation at the end of next cycle
busy1:  mov dx, INCSR1          ;poll BUSY bit til operation completed
        in  ax, dx
        bt  ax, 7
        jc  short busy1

;program the output look-up table if requested
;Display prompt message for input:

pal:    mov ax, 21h
        mov PINTFRAME.VMINT,eax
        mov edx, OFFSET palette
        mov ah, 9
        mov ebx, OFFSET PINTFRAME
        VM86CALL

;read yes (y) or no (n) answer

answer: mov ax, 21h
        mov PINTFRAME.VMINT,eax
        mov ah, 07h                     ;read key, return value in al
        mov ebx, OFFSET PINTFRAME
        VM86CALL

        mov bl, al
        cmp al, 'y'
        je  luts
        mov al, bl
        cmp al, 'n'
        je  load
        jmp pal

;Beginning of Output LUT Programming

luts:

;save board operation mode

        mov dx, OUTCSR
        in  ax, dx
        push ax
        mov dx, INCSR2
        in  ax, dx
        push ax

;set board for LUT programming

        mov ax, LDINLUT             ;select MODE=010
        out dx, ax

;load up output LUT

        mov ecx, 00h
lut:    mov dx, OUTCSR              ;select output LUT 0
        mov ax, OSEL
        out dx, ax
        mov dx, INDEX                ;select index into LUT 0
        mov ax, cx
        out dx, ax
        mov edi, ecx
        mov ebx, OFFSET red
        mov al, BYTE PTR ds:[ebx+edi]
        mov ebx, OFFSET green
        mov ah, BYTE PTR ds:[ebx+edi]
        mov dx, RGLUT
        out dx, ax
        mov ebx, OFFSET blue
        mov al, BYTE PTR ds:[ebx+edi]
        mov dx, BLUT
        out dx, ax
        inc ecx
        cmp ecx, 0FFh
        jle lut

;restore board original operating mode
        pop ax
        mov dx, INCSR2
        out dx, ax
        pop ax
        mov dx, OUTCSR
        out dx, ax
;End of LUT programming - show the result:

        mov dx, OUTCSR      ;set active bus buffers to buff 2 and 3
        mov ax, BUS23
        out dx, ax

        mov ecx, 512
        mov eax, buff0       ;fill buff 2 with pattern
        xor edi, edi
        xor edx, edx
fill:   mov BYTE PTR fs:[eax], dl
        inc eax
        mov BYTE PTR fs:[eax], dl
        inc eax
        inc edi
        cmp edi, 512
        jl short endlut
        xor edi, edi
        inc edx
        dec ecx
endlut: jnz short fill

        mov dx, OUTCSR              ;set active bus buffers to buff 0 and 1
        mov ax, BUS01
        out dx, ax

;define the input buffer's active window

load:   mov dx, STARTR
        mov ax, SLINE              ;set buffer start line at 0
        out dx, ax
        mov dx, ENDR                ;set buffer end line at 360
        mov ax, ELINE              ;this is enough data for 6 degree field
        out dx, ax                  ;for both lead and lag array

;set operating mode

        mov dx, INCSR1
        mov ax, ALU                ;set ALU values at 0000, NO CARRY
        out dx, ax                 ;keep BUSY and PASS clear

        mov dx, OUTCSR
        in  ax, dx
        btr ax, 6                   ;clear TRGEN bit.
        out dx, ax

;input offset angle count

;Display prompt message for input:
reset:  mov ax, 21h
        mov PINTFRAME.VMINT,eax
        mov edx, OFFSET rotmsg
        mov ah, 9
        mov ebx, OFFSET PINTFRAME
        VM86CALL

;read 4 input characters + CR and store in keybuf buffer
rd_buf: mov ax, 21h
        mov PINTFRAME.VMINT, eax
        mov edx, OFFSET keybuf
        mov ah, 0Ah
        mov ebx, OFFSET PINTFRAME
        VM86CALL

;convert ASCII to hex and place in ECX register
        xor ecx, ecx
        xor edx, edx
        mov eax, OFFSET msg
        mov bx, WORD PTR [eax]
        sub bh, 30h
        mov bl, bh
        and bx, 00FFh
        cmp bx, 9
        jle  skiph1
        sub bl, 7
skiph1: mov cl, bl
        shl ecx, 4
        add eax, 1
        mov bx, WORD PTR [eax]
        sub bh, 30h
        mov bl, bh
        and bx, 00FFh
        cmp bx, 9
        jle  skiph2
        sub bl, 7
skiph2: add cl, bl
        shl ecx, 8
        mov edx, ecx
        xor ecx, ecx
        add eax, 1
        mov bx, WORD PTR [eax]
        sub bh, 30h
        mov bl, bh
        and bx, 00FFh
        cmp bx, 9
        jle  skiph3
        sub bl, 7
skiph3: mov cl, bl
        shl ecx, 4
```

```
         add eax, 1
         mov bx, WORD PTR [eax]
         sub bh, 30h
         mov bl, bh
         and bx, 00FFh
         cmp bx, 9
         jle  skiph4
         sub bl, 7
skiph4:  add cl, bl
         add ecx, edx
         push ecx

;check for arrows key input
;inclement counter if right arrow
;decrement counter if left arrow

nextfrm:        ; entry point for next frame load

;check if arrow key was pressed (check status of keyboard buffer)

ck_key: mov ax, 21h
         mov PINTFRAME.VMINT, eax
         mov ah, 0Bh                    ;check keyboard input status
         mov ebx, OFFSET PINTFRAME
         VM86CALL                       ;set al to zero if no key pressed
         cmp al, 00h
         je  nokey
rd_key: mov ax, 21h                     ;read key if status set
         mov PINTFRAME.VMINT, eax
         mov ah, 07h                    ;read key, return value in al
         mov ebx, OFFSET PINTFRAME
         VM86CALL
         mov ax, 21h                    ;repeat function to get extended code
         mov PINTFRAME.VMINT, eax
         mov ah, 07h                    ;read key, return value in al
         mov ebx, OFFSET PINTFRAME
         VM86CALL
         cmp al, 46h                    ;check for Ctl-Break
         je quit
         cmp al, 4Fh                    ;check for End Key
         je quit
         cmp al, 47h                    ;check for Home Key
         je home
         cmp al, 4Bh                    ;check for left arrow
         je  left
         cmp al, 4Dh                    ;check for right arrow
```

```
        je  right
        jmp nokey
home:   JMP32S reset
left:   pop ecx
        mov ax, rotate
        sub cx, ax
        push ecx
        jmp nokey
right:  pop ecx
        mov ax, rotate
        add cx, ax
        push ecx
nokey:
        pop ecx
        push ecx

;Display current sector count offset
;First convert Hex to ASCII

        mov ax, cx
        and al, 00001111b
        cmp al, 09h
        jle skipa1
        add al, 37h
        jmp disp1
skipa1: add al, 30h
disp1:  mov sector[3], al
        mov ax, cx
        shr ax, 4
        and al, 00001111b
        cmp al, 09h
        jle skipa2
        add al, 37h
        jmp disp2
skipa2: add al, 30h
disp2:  mov sector[2], al
        mov ax, cx
        shr ax, 8
        and al, 00001111b
        cmp al, 09h
        jle skipa3
        add al, 37h
        jmp disp3
skipa3: add al, 30h
disp3:  mov sector[1], al
        mov ax, cx
        shr ax, 12
        and al, 00001111b
        cmp al, 09h
        jle skipa4
        add al, 37h
        jmp disp4
skipa4: add al, 30h
disp4:  CALL32F sel_code32, CLS    ;clear PC screen
        mov sector, al
        mov ax, 21h
        mov PINTFRAME.VMINT,eax
        mov edx, OFFSET sectmsg
        mov ah, 9
        mov ebx, OFFSET PINTFRAME
        VM86CALL

;write offset angle to JDR port

        mov dx, JDR            ;set delay counter
        mov ax, cx
        out dx, ax

;load up one frame in buffer 0

        mov dx, INCSR2         ;set MODE for External Port Input
        mov ax, EPORTI         ;MODE=111, Input Buffer 0
        out dx, ax.

        mov dx, INCSR1
        mov ax, LOADBUF        ;set BUSY. keep PASS clear
        out dx, ax             ;start of load operation

poll:   mov dx, INCSR1         ;wait til load completed, poll BUSY bit
        in  ax, dx
        bt  ax, 7
        jc  short poll

;initialize pointers for the sort routine

        xor eax, eax
        xor edi, edi
        xor ecx, ecx

;start unscrambling

;3 degree image:
        mov ebx, buff0                  ;load base address of buffer 0
sort3:  mov dl, BYTE PTR fs:[ebx]       ;load scrambled byte from buffer 0
        mov ebp, OFFSET order           ;lookup offset value from order array
        mov al, BYTE PTR ds:[ebp+edi]   ;load offset value from order table
        shl eax, 9                      ;multiply row by 512 pixels/row
        mov ebp, buff1                  ;load base address of buffer 1
        add eax, ebp                    ;offset to unscrambled row
        add eax, ecx                    ;offset to unscrambled column
        mov BYTE PTR fs:[eax], dl       ;write byte at unscrambled linear address

;update pointers
        xor eax, eax                    ;clear eax
        inc ebx                         ;increment byte pointer in buffer 0
        inc edi                         ;increment offset pointer in table order
        cmp edi, 180                    ;reached row 180?
        jl  short skip1                 ;no, skip
        xor edi, edi                    ;yes, reset pointer at zero and
        inc ecx                         ;start new column
skip1:  cmp ecx, 512                    ;last column?
        jl short sort3                  ;no, continue

;6 degree image:

        xor eax, eax
        xor edi, edi
        xor ecx, ecx
sort6:  mov dl, BYTE PTR fs:[ebx]       ;load scrambled byte from buffer 0
        mov ebp, OFFSET order           ;lookup offset value from order array
        mov al, BYTE PTR ds:[ebp+edi]   ;load offset value from order table
        shl eax, 9                      ;multiply row by 512 pixels/row
        mov ebp, buff1                  ;load base address of buffer 1
        add eax, ebp                    ;offset to unscrambled row
        add eax, 92160                  ;offset to second half of buffer
        add eax, ecx                    ;offset to unscrambled column
        mov BYTE PTR fs:[eax], dl       ;write byte at unscrambled linear address

;update pointers
        xor eax, eax                    ;clear eax
        inc ebx                         ;increment byte pointer in buffer 0
        inc edi                         ;increment offset pointer in table order
        cmp edi, 180                    ;reached row 180?
        jl  short skip2                 ;no, skip
        xor edi, edi                    ;yes, reset pointer at zero and
        inc ecx                         ;start new column
skip2:  cmp ecx, 512                    ;last column?
```

```
        jl short sort6                ;no, continue



;split up lead and lag arrays and expand each row
;by a factor of five. expanded image ends up in buffer 0

lead:   mov ebx, buff1      ;pointer to beginning of lead array

;comment out next line if lead array is wanted
lag:    add ebx, 46080      ;pointer to beginning of lag array
        mov eax, buff0
        xor edi, edi
        xor esi, esi

        mov ecx, 256
expand: mov dl, BYTE PTR fs:[ebx]
        mov BYTE PTR fs:[eax], dl
        add eax, 512
        mov BYTE PTR fs:[eax], dl
        add eax, 512
        mov BYTE PTR fs:[eax], dl
        add eax, 512
        mov BYTE PTR fs:[eax], dl
        add eax, 512
        mov BYTE PTR fs:[eax], dl
        add ebx, 92160
        sub eax, 1792
        mov dl, BYTE PTR fs:[ebx]
        mov BYTE PTR fs:[eax], dl
        add eax, 512
        mov BYTE PTR fs:[eax], dl
        add eax, 512
        mov BYTE PTR fs:[eax], dl
        add eax, 512
        mov BYTE PTR fs:[eax], dl
        add eax, 512
        mov BYTE PTR fs:[eax], dl
        sub ebx, 92158
        sub eax, 2303
        loopne expand

;loop control

        mov ecx, 256
        inc edi
        mov edx, edi
        shl edx, 9
        imul edx, 5
        mov eax, buff0
        add eax, edx
        cmp edi, 90
        jl short expand

;branch back for lead array (have yet to figure out a way!!)
;for the moment this does lag array only!!



;remove third A/D converter offset

        mov ebx, buff0
        mov eax, s_row
        shl eax, 9
        mov ecx, height
        shl ecx, 7
        add ebx, eax
dc_off: mov eax, DWORD PTR fs:[ebx]
        sub eax, 03030303h
        mov DWORD PTR fs:[ebx], eax
        dec ecx    .
        jecxz skip3
        add ebx, 04h
        jmp dc_off
skip3:

;copy final image in display buffer 2

        mov ebx, buff0              ;source buffer 0
        mov edi, 57600
pipe:   mov dx, OUTCSR
        mov ax, BUS01               ;set active bus buffers 0 and 1
        out dx, ax
        mov ecx, DWORD PTR fs:[ebx]
        mov dx, OUTCSR
        mov ax, BUS23               ;set active bus buffers 2 and 3
        out dx, ax
        mov DWORD PTR fs:[ebx], ecx ;buffer 2 is now new base address
        add ebx,4              ;at 0A00000h
        add ebp,4
        mov ecx, edi
        sub ecx, 01h
        jecxz  short skip4
        mov edi, ecx
        jmp short pipe
skip4:
        mov ax, BUS01         ;restore buffer 0 at base address 0A00000h
        mov dx, OUTCSR
        out dx, ax

busy2:  mov dx, INCSR1         ;poll BUSY bit til operation completed
        in  ax, dx .
        bt  ax, 7
        jc  short busy2

;Perform absolute 16 bit jump (in a 16 bit segment)
        JMP32S nextfrm

;End of program
quit:   mov ax, 21h
        mov PINTFRAME.VMINT,eax
        mov edx, OFFSET endmsg     ;display end of prog message
        mov ah, 9
        mov ebx, OFFSET PINTFRAME
        VM86CALL

        pop ecx                    ;clean up stack
        popad                      ;restore all 32-bit registers from stack
        BACK2DOS
USER   ENDP
        PROT_CODE_END
```

DT-2861.INC INCLUDE FILE

```
;DT-2861 registers

INCSR1      equ    0250h
INCSR2      equ    0252h
OUTCSR      equ    0254h
INDEX       equ    0258h
INLUT       equ    025Ah
RLUT        equ    025Ah
RGLUT       equ    025Ch
BLUT        equ    025Eh
OUTCSR      equ    0254h
STARTR      equ    025Ch
ENDR        equ    025Eh
YPAN        equ    025Ah
XPAN        equ    0258h

        ALIGN  4
buff0  equ    0A00000h   ;base addresses of frame buffers 0 to 15
buff1  equ    0A40000h
```

ORDER.INC INCLUDE FILE

```
;order look-up table lists the scramble order
;of each block of 180 consecutive bytes in buffer 1
        ALIGN   1
order   db  2,17,32,47,62,77, 92,107,122,137,152,167
        db  3,18,33,48,63,78, 93,108,123,138,153,168
        db  4,19,34,49,64,79, 94,109,124,139,154,169
        db  5,20,35,50,65,80, 95,110,125,140,155,170
        db  6,21,36,51,66,81, 96,111,126,141,156,171
        db  7,22,37,52,67,82, 97,112,127,142,157,172
        db  8,23,38,53,68,83, 98,113,128,143,158,173
        db  9,24,39,54,69,84, 99,114,129,144,159,174
        db 10,25,40,55,70,85,100,115,130,145,160,175
        db 11,26,41,56,71,86,101,116,131,146,161,176
        db 12,27,42,57,72,87,102,117,132,147,162,177
        db 13,28,43,58,73,88,103,118,133,148,163,178
        db 14,29,44,59,74,89,104,119,134,149,179,164
        db  0,15,30,45,60,75, 90,105,120,135,150,165
        db  1,16,31,46,61,76, 91,106,121,136,151,166
```

LUT.INC INCLUDE FILE

```
;Output LUT entries.
;Values in all three tables: blue, green and red range from 0 to 255 or 00h to 0FFh

blue   db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;10
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;20
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;30
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;40
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;50
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;60
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;70
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;80
       db   00h, 00h, 00h, 00h, 00h, 05h, 0Ah, 10h, 15h, 1Ah                ;90
       db   20h, 25h, 2Ah, 30h, 35h, 3Ah, 40h, 45h, 4Ah, 50h                ;100
       db   55h, 5Ah, 60h, 65h, 6Ah, 70h, 75h, 7Ah, 80h, 85h                ;110
       db   8Ah, 90h, 95h, 9Ah,0A0h,0A5h,0AAh,0B0h,0B5h,0BAh                ;120
       db   0C0h,0C5h,0CAh,0D0h,0D5h,0DAh,0E0h,0E5h,0EAh,0F0h               ;130
       db   0F5h,0FAh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh               ;140
       db   0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH               ;150
       db   0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH               ;160
       db   0F0h,0E0h,0D0h,0C0h,0B0h,0A0h,090h,080h,070h,060h               ;170
       db   50h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h                ;180
       db   40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h                ;190
       db   40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h                ;200
       db   40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h                ;210
       db   40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h                ;220
       db   40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h                ;230
       db   40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h                ;240
       db   40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h, 40h                ;250
       db   40h, 40h, 40h, 40h, 40h, 40H                                    ;256

green  db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;10
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;20
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;30
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;40
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;50
       db   00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h                ;60
       db   05h, 0Ah, 10h, 15h, 1Ah, 20h, 25h, 2Ah, 30h, 35h                ;70
       db   3Ah, 40h, 45h, 4Ah, 50h, 55h, 5Ah, 60h, 65h, 6Ah                ;80
       db   70h, 75h, 7Ah, 80h, 85h, 8Ah, 90h, 95h, 9Ah,0A0h                ;90
       db   0A5h,0AAh,0B0h,0B5h,0BAh,0C0h,0C5h,0CAh,0D0h,0D5h               ;100
       db   0DAh,0E0h,0E5h,0EAh,0F0h,0F5h,0FAh,0FFh,0FFh,0FFh               ;110
       db   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh               ;120
       db   0FAh,0F5h,0F0h,0EAh,0E5h,0E0h,0DAh,0D5h,0D0h,0D0h               ;130
       db   0CAh,0C5h,0C0h,0BAh,0B5h,0B0h,0AAh,0A5h,0A0h, 9Ah               ;140
```

```
        db      95h, 90h, 8Ah, 85h, 80h, 7Ah, 75h, 70h, 6Ah, 65h            ;150
        db      60h, 5Ah, 55h, 50h, 4Ah, 45h, 40h, 3Ah, 35h, 30h            ;160
        db      2Ah, 25h, 20h, 1Ah, 15h, 10h, 0Ah, 05h, 00h, 00h            ;170
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;180
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;190
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;200
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;210
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;220
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;230
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;240
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;250
        db      00h, 00h, 00h, 00h, 00h, 00H                                ;256

red     db      30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h            ;10
        db      30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h            ;20
        db      30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h            ;30
        db      30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h, 30h            ;40
        db      30h, 31h, 32h, 33h, 34h, 35h, 36h, 37h, 38h, 39h            ;50
        db      3Ah, 3Bh, 3Ch, 3Dh, 3Eh, 3Fh, 42h, 44h, 46h, 48h            ;60
        db      4Ah, 4Ch, 4Eh, 53h, 56h, 59h, 5Dh, 64h, 69h,070h            ;70
        db      078h,080h,088h,08Bh,08Eh,092h,095h,098h,09Bh,09Eh           ;80
        db      0A2h,0A8h,0AAh,0B0h,0B5h,0BAh,0C0h,0C3h,0C6h,0C9h           ;90
        db      0CBh,0CEh,0D2h,0D5h,0DAh,0E0h,0E5h,0EAh,0F0h,0F5h           ;100
        db      0FAh,0FFh,0FFh,0F0h,0E0h,0D0h,0C0h,0B0h,0A0h,090h           ;110
        db      080h,070h, 60h, 50h, 40h, 30h, 20h, 10h, 00h, 00h           ;120
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;130
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;140
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;150
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;160
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;170
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;180
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;190
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;200
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;210
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;220
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;230
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;240
        db      00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h            ;250
        db      00h, 00h, 00h, 00h, 00h, 00h                                ;256
```

LIST OF REFERENCES

1. Ayers, G.R. "Calibration and Initialization of the IRSTD System", MS Thesis, Dec.1987.

2. Engel, R.C. "A PC Based Imaging System for the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System", MS Thesis, Sept. 1989.

3. Baca, M.J. "Real Time Imaging of Infrared Background Scene Data Generated by the Naval Postgraduate School Infrared Search and Target Designation (NPS-IRSTD) System", MS Thesis, Sept 1990.

4. E.C. Crittenden Jr, A.W. Cooper "Modification, Testing and Calibration of Infrared Search and Target Designation Hardware Received from NSWC", NPS Report NPS61-89-011CR, May 1989.

5. Cooper, A.W. W.J. Lentz, R.C. Engel; "Background Measurements using the NPS-IRST System", SPIE Proceedings Vol. 1311, "Characterization, Propagation and Simulation of Infrared Scenes", 1990.

6. Cooper, A.W., W.J. Lentz, M.J. Baca and J.D. Bernier, "Image Display and Background Analysis with the Naval Postgraduate School Infrared Search and Track System", SPIE Proceedings, Vol. 1846, 1991.

7. Cooper, A.W., W.J. Lentz, E.D. Bloedel, L. Yee, R.R. Keever, M.A. Polehn, B.E. Northon, R.N. Murrata; "Joint Naval Postgraduate School and Boeing Aerospace Company AN/SAR-8 IRST Measurements, Experimental Results 20-22 Sept 1989.

8. Cooper, A.W. NACIT Proposal for Research for period 1 October 1991 to 30 September 1992. Naval Sea Systems Command, PMS-421.

9. Parker, G. "Development of Modified Detector/DEWAR Package for the ADM IRSTD and Analysis of the Effect of the Optical Performance of the System Using the SuperOslo Computer Code", MS Thesis, Sept. 1986

10. R.C. Gonzalez, P. Wintz, "Digital Image Processing", 2nd ed., Addison-Wesley Publishing Company, 1987.

11. IRIS Tutor User Manual, 3rd ed., Data Translation Inc., Marlboro, MA, 1987.

12. DT-IRIS User Manual, 3rd ed., Data Translation Inc., Marlboro, MA, 1988.

13. User Manual for the DT-2861 High Speed Arithmetic Frame Grabber, 1st ed., Data Translation Inc. Marlboro, MA, 1987.

14. 386 DX "Microprocessor Programmer's Reference Manual", Intel Corporation, Osborne McGraw-Hill, 1989.

15. J.H. Crawford, P.P. Gelsinger, "Programming the 80386", Sybex Inc., 1987.

16. J.L. Turley "Advanced 80386 Programming Techniques", McGraw-Hill, Berkeley, CA, 1988.

17. Neal Margulis, "80386 Protected Mode Initialization", Doctor Dobb's Journal, October 1988.

18. Al Williams, "DOS 5: A Developer's Guide", Advanced Programming Guide to DOS,M&T Publishing, Inc., August 1991.

19. Al Williams, "Roll Your Own DOS Extender:Part 1", Doctor Dobb's Journal, October 1990.

20. Al Williams, "Roll Your Own DOS Extender:Part 2", Doctor Dobb's Journal, November 1990.

21. Neal Margulis, "Advanced 80386 Memory Management", Doctor Dobb's Journal, April 1989.

22. Alan R. Miller, DOS Assembly Language Programming, Sybex Inc., 1988.

23. Michael J. Young, "Inside DOS:A Programmer's Guide", Sybex Inc., 1988.

24. Microsoft Macro Assembler Version 5.1, Macro Assembler for the MS-DOS Operating System, Programmer's Guide, Microsoft Corporation, 1987.

25. Paul and Meg Noah, VGAIPS User's Manual, Disk file Accompanying the VGAIPS Program.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Chairman, Code PH Department of Physics Naval Postgraduate School Monterey, CA 93943-5000	1
5.	Professor A.W. Cooper, Code PH/Cr Department of Physics Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Arthur Cote, Jr., PMS-421 Naval Sea Systems Command Washington, DC 20362-5101	1
7.	S.K. Petropoulous, Code R42 White Oak Laboratory Naval Surface Weapons Center Detachment Silver Springs, MD 20903-5000	1
8.	M.W. Zurasky, Code 6552 Naval Surface Weapons Center Dahlgren, VA 22448-5000	1
9.	Robert J.L. Corriveau Director, Electro-Optics Division Defence Research Establishment Valcartier 2459 Pie XI Blvd., North (P.O. Box 8800) Courcelette, Quebec, Canada, G0A 1R0	1

10. Dr. Louis G. Gregoris 1
Spar Aerospace Limited
Director, Electro-Optical Technology
1235 Ormont Drive
Weston, Ontario, Canada, M9L 2W6

11. R. Bloedel 1
Manager, Signal Processing Technology
Boeing Company
P.O. Box 3999
Seattle, WA 98124-2499

12. DLAEEM 4-3 Maj. J.D. Bernier 1
National Defence Headquarters
Major General R. Perkes Building
Ottawa, Ontario, Canada, KA1 0K2

497-769